

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

“UALTools -  
Herramientas  
contenerizadas para  
construcción, testeo y  
desarrollo de  
aplicaciones”

Curso 2018/2019

**Alumno/a:**

David Vicente Ocaña Robles

**Director/es:**

José Joaquín Cañadas Martínez

## AGRADECIMIENTOS:

A mi familia, la que se elige y la que no, a Rocío, por haberlo hecho todo más sencillo, a Altiplaconsulting, por haberme formado laboralmente y haber conseguido ilusionarme, a la docencia que ha sabido guiarme, en especial, a Fco Gabriel Guil Reyes, por inspirar mi camino en los comienzos, a José Fernando Bienvenido Bárcena, por haber conseguido exponer mi potencial al máximo y a José Joaquín Cañadas Martínez, por mostrarme el punto de vista del desarrollo de software más fiel a la realidad actual y por acompañarme en esta última etapa, por supuesto, no olvidar a mis compañeros, por estar en las buenas y en las malas y por último a la Universidad de Almería, por haber sido mi segundo hogar y haberme hecho ver la luz, a la ciencia y al progreso.



# Índice de contenidos

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Motivación . . . . .	1
1.2	Objetivos . . . . .	1
1.2.1	Objetivos formativos . . . . .	1
1.2.2	Objetivos de la aplicación . . . . .	2
1.2.3	Objetivos personales . . . . .	3
1.3	Planificación . . . . .	3
1.3.1	Organización y seguimiento - Trello . . . . .	4
1.4	GitHub . . . . .	5
1.4.1	Análisis de herramientas . . . . .	5
1.4.2	Diseño UALTools . . . . .	5
1.4.3	Implementación UALTools & Pruebas UALTools . . . . .	6
1.4.4	Redacción del TFG . . . . .	6
1.5	Estructura de la memoria . . . . .	6
<b>2</b>	<b>Estado del arte</b>	<b>9</b>
2.1	XAMPP . . . . .	9
2.2	Docker compose . . . . .	10
2.3	Comparativa . . . . .	11
<b>3</b>	<b>Tecnologías y Herramientas</b>	<b>13</b>
3.1	Ubuntu MATE . . . . .	13
3.2	Postman . . . . .	14
3.3	Trello . . . . .	14
3.4	Go . . . . .	15
3.5	Docker . . . . .	15
3.5.1	¿Qué diferencia a los contenedores de una máquina virtual?	15
3.6	Docker compose . . . . .	17
3.7	Sublime Text . . . . .	17
3.8	Git . . . . .	18
3.9	GitHub . . . . .	18
3.10	Cobra . . . . .	19
3.11	GCloud . . . . .	19
<b>4</b>	<b>UALTools</b>	<b>21</b>
4.1	Requisitos del sistema . . . . .	21
4.1.1	Requisitos generales . . . . .	21
4.1.2	Requisitos no funcionales . . . . .	23
4.2	Diagrama de casos de uso . . . . .	24
4.2.1	Especificación de actores del sistema . . . . .	25
4.2.2	Especificación de casos de uso del sistema . . . . .	25
4.3	Arquitectura . . . . .	27
4.4	Implementación . . . . .	28
4.5	Controladores de Docker . . . . .	29

4.5.1	pkg/docker/container.go . . . . .	29
4.5.2	pkg/docker/image.go . . . . .	33
4.5.3	pkg/docker/network.go . . . . .	33
4.5.4	pkg/docker/container_options.go . . . . .	35
4.5.5	pkg/docker/watcher.go . . . . .	37
4.6	Especificación de contenedores . . . . .	41
4.7	Ejecución de contenedores . . . . .	43
4.8	Configuración de la aplicación . . . . .	45
4.8.1	pkg/config/config.go . . . . .	45
4.8.2	pkg/config/env.go . . . . .	47
4.8.3	pkg/config/project.go . . . . .	48
4.9	Contenedores base de UALTools . . . . .	49
4.9.1	dev-go . . . . .	50
4.9.2	dev-java . . . . .	51
4.9.3	go . . . . .	52
4.9.4	java . . . . .	52
4.9.5	python . . . . .	52
4.9.6	mysql . . . . .	53
4.9.7	redis . . . . .	53
4.9.8	phpmyadmin . . . . .	53
4.9.9	migrator . . . . .	53
4.10	Comandos . . . . .	54
4.10.1	Librería <a href="https://github.com/spf13/cobra">https://github.com/spf13/cobra</a> . . . . .	54
4.10.2	main.go . . . . .	55
4.10.3	cmd_root.go . . . . .	55
4.10.4	cmd_start.go . . . . .	56
4.10.5	cmd_stop.go . . . . .	61
4.10.6	cmd_restart.go . . . . .	62
4.10.7	cmd_rm.go . . . . .	62
4.10.8	cmd_tool.go . . . . .	64
4.10.9	cmd_cache_print.go . . . . .	66
4.10.10	cmd_debug.go . . . . .	66
4.10.11	cmd_run.go . . . . .	67
4.10.12	cmd_pull.go . . . . .	68
4.11	Docker compose y pruebas del entorno . . . . .	69
4.12	Despliegue . . . . .	71
<b>5</b>	<b>Resultados</b> . . . . .	<b>75</b>
5.1	Ejemplo de uso de herramientas de UALTools . . . . .	75
5.1.1	Ejecución de scripts de Python . . . . .	75
5.1.2	Ejecución de test JUnit de Java . . . . .	75
5.1.3	Ejecución de comandos desde dentro de un contenedor . . . . .	77
5.2	Ejecución de un entorno de desarrollo con distintos servicios de UALTools . . . . .	78
5.2.1	Estructura del proyecto de ejemplo . . . . .	78
5.2.2	Ejecución de migraciones . . . . .	80

5.2.3	Ejemplo API REST con Java y MySQL . . . . .	82
5.2.4	Ejemplo API REST en Go con Redis . . . . .	86
<b>6</b>	<b>Conclusiones</b>	<b>89</b>
6.1	Trabajo futuro . . . . .	89



## Índice de figuras

1	Representación de entornos de UALTools. . . . .	3
2	Planificación temporal. . . . .	3
3	Panel Trello UALTools. . . . .	4
4	Progreso de commits en GitHub. . . . .	5
5	Logo de XAMPP. . . . .	9
6	Logo de Apache. . . . .	9
7	Logo de mariadb. . . . .	10
8	Logo de PHP. . . . .	10
9	Logo de Perl. . . . .	10
10	Logo de docker-compose. . . . .	10
11	Tabla comparativa de herramientas similares. . . . .	11
12	Logo de Ubuntu MATE. . . . .	13
13	Logo de Postman. . . . .	14
14	Logo de trello. . . . .	14
15	Logo de Go. . . . .	15
16	Logo de Docker. . . . .	15
17	Virtualización de máquinas virtuales. . . . .	16
18	Virtualización de contenedores. . . . .	16
19	Sublime Text logo. . . . .	17
20	Git logo. . . . .	18
21	GitHub logo. . . . .	18
22	Logo Cobra. . . . .	19
23	Google Cloud Platform logo. . . . .	19
24	Diagrama casos de uso . . . . .	24
25	Arquitectura UALTools. . . . .	27
26	Árbol del proyecto UALTools. . . . .	28
27	Árbol de ficheros pkg/docker. . . . .	29
28	Gestor de contenedores Docker. . . . .	30
29	Constructor de contenedores. . . . .	31
30	Constructor de imágenes. . . . .	33
31	Constructor de redes. . . . .	34
32	Opción funcional WithImage. . . . .	35
33	Tipo ContainerOption. . . . .	35
34	Ejecución de opciones funcionales. . . . .	36
35	Definición y constructor de Watcher. . . . .	37
36	Métodos Run y Wait del watcher. . . . .	38
37	Método privado runForeground. . . . .	39
38	Personalización y logging de las herramientas y servicios. . . . .	40
39	Comienzo del código fuente de <b>pkg/containers/containers.go</b> . . . . .	41
40	Función de ejecución de comandos Interactive. . . . .	44
41	Función de ejecución de comandos InteractiveWithOutput. . . . .	45
42	Árbol de directorio de configuración. . . . .	45
43	Inicio del fichero pkg/config/config.go. . . . .	46



44	Servicios y herramientas de la configuración. . . . .	46
45	Fichero de comprobación de variables de entorno. . . . .	48
46	Fichero de configuración pkg/config/project.go. . . . .	49
47	Árbol de contenedores de la aplicación. . . . .	49
48	Dockerfile del contenedor dev-go. . . . .	50
49	Script run.sh que se ejecuta al poner en marcha el contenedor. . .	51
50	Dockerfile del contenedor dev-java. . . . .	51
51	Dockerfile del contenedor go. . . . .	52
52	Dockerfile del contenedor java. . . . .	52
53	Dockerfile del contenedor python. . . . .	52
54	Dockerfile del contenedor mysql. . . . .	53
55	Dockerfile del contenedor Redis. . . . .	53
56	Dockerfile del contenedor phpmyadmin. . . . .	53
57	Dockerfile del contenedor migrator. . . . .	53
58	Árbol de ficheros del directorio <b>cmd/ualtools</b> . . . . .	54
59	Punto de entrada de la aplicación. . . . .	55
60	Raíz de ejecución de comandos. . . . .	55
61	Inicio de fichero de comando start. . . . .	56
62	Método privado <b>resolveDeps</b> . . . . .	57
63	Preparación de herramientas. . . . .	58
64	Creación de watcher y preparación de servicios. . . . .	60
65	Fichero de comando para parar servicios. . . . .	61
66	Fichero de comando para reiniciar servicios. . . . .	62
67	Fichero de comando para eliminar contenedores de servicios. . . .	63
68	Método para crear el entrypoint de una herramienta. . . . .	64
69	Fichero de comando para ejecutar herramientas en directorios. . .	65
70	Fichero de comando para mostrar directorio de caché de herramientas.	66
71	Fichero de comando para activar logging de depuración. . . . .	66
72	Fichero de comando para acceder a contenedores para ejecutar comandos. . . . .	67
73	Fichero de comando para actualizar imágenes de contenedores. . .	68
74	Fichero docker-compose del proyecto UALTools. . . . .	69
75	Ejecución comando ‘docker-compose build’. . . . .	70
76	Compilación del programa UALTools con Go. . . . .	71
77	Ejecución del programa UALTools. . . . .	71
78	Características Bucket de Storage para subida de binarios. . . . .	72
79	Binario Linux subido a Storage . . . . .	73
80	Script de subida de contenedores. . . . .	73
81	Imágenes en container registry. . . . .	74
82	hello.py. . . . .	75
83	Hello World en python. . . . .	75
84	Fichero Abc.java . . . . .	76
85	Fichero AbcTest.java . . . . .	76
86	Directorio javaexample. . . . .	76
87	Directorio javaexample con ficheros compilados. . . . .	77
88	Ejecución test JUnit con UALTools. . . . .	77

89	Ejecución comandos dentro de contenedor Python. . . . .	77
90	Directorio example. . . . .	78
91	ualtools.yml . . . . .	79
92	Makefile . . . . .	80
93	Schema de migrator. . . . .	81
94	Ficheros de migración. . . . .	81
95	Creación de base de datos example. . . . .	81
96	Creación de tabla users. . . . .	82
97	Ejecución comando make migrations. . . . .	82
98	Tabla de usuarios creada vía migraciones. . . . .	82
99	Ejecución del servicio javaenv. . . . .	83
100	Log Spring del servicio javaenv. . . . .	83
101	API de usuarios en Java. . . . .	84
102	Creación de un usuario con la API de Java. . . . .	85
103	Usuario creado en la base de datos. . . . .	85
104	Ejecución del servicio de desarrollo goenv. . . . .	86
105	Ejecución del servicio de desarrollo goenv. . . . .	86
106	Creación de un par nombre-trabajo. . . . .	87
107	Visualización del par nombre-trabajo creado anteriormente. . . .	87



## RESUMEN

UALTools es una herramienta de creación de entornos de desarrollo en Go y Java, que permite la ejecución de scripts o test unitarios básicos en Java, Python y Go, permite crear bases de datos en MySQL y Redis y que además posibilita el crear bases de datos MySQL a través de la ejecución de migraciones. UALTools es sencillo de instalar (sólo requiere la instalación de Docker) y configurar (se configura con un fichero **ualtools.yml**). Proporciona acceso a un set de herramientas que permite ejecutar programas o scripts de prueba de la forma más sencilla posible, configurando únicamente cosas esenciales de cada una de las herramientas involucradas en el proceso.

## ABSTRACT

UALTools is a tool for creating development environments in Go and Java, which enables the execution of basic scripts or unit tests in Java, Python and Go, allows creating databases in MySQL and Redis and also makes it possible to create databases MySQL through the execution of migrations. UALTools is easy to install (only requires the installation of Docker) and configure (it is configured with a file **ualtools.yml**). It provides access to a set of tools that allows you to run programs or unit tests in the simplest way possible, configuring only essential things of each of the tools involved in the process.



# 1 Introducción

En el presente Trabajo de Fin de Grado (**TFG** en adelante), se hará una descripción del proceso de investigación y desarrollo de la aplicación UALTools, creada por David Vicente Ocaña Robles (**el autor** en adelante). Dicha aplicación se puede describir como una aplicación de consola, para la creación de entornos de desarrollo y ejecución de scripts en distintos lenguajes de programación, basada en contenedores Docker y compatible con distintos sistemas operativos. La aplicación estará desarrollada en Go y con librerías de software libre.

## 1.1 Motivación

La razón principal que inspira el desarrollo del presente TFG, es la de brindar la posibilidad a los usuarios más inexpertos, estudiantes de ingeniería informática y otras ingenierías, de agilizar el proceso de desarrollo de software de un entorno de programación, sin necesidad alguna de instalar (hablando en términos de tamaño de datos) pesados IDEs (como Visual Studio, Eclipse o IntelliJ) y posteriormente configurarlos. Una ardua tarea cuando sobre todo, no se está familiarizado con este tipo de software, y que en muchos casos, puede llevar al usuario a perder horas cruciales y valiosas al principio de su aprendizaje.

Otra de las razones que inspira el desarrollo del presente TFG, es la de mostrar un enfoque sencillo de la ejecución de aplicaciones a través del Terminal (Linux, MacOS) o Consola (Windows), haciendo de éstos una herramienta esencial para todo el proceso de desarrollo y puesta en marcha de una aplicación. La principal razón de poner en valor este enfoque es que en el mundo laboral, en muchas ocasiones es algo con lo que hay que convivir día a día.

Por último, se pretenden mostrar las inquietudes del autor de añadir una capa de abstracción en cuanto al uso de Docker [5] reduciendo su complejidad y aumentando su comprensión, elaborando comandos que simplifican a lo esencial la utilización de dicha herramienta y por tanto, haciendo que UALTools sea un primer paso perfecto para la comprensión, sintetización e introducción a Docker y otras herramientas como Docker-Compose [1].

## 1.2 Objetivos

En el presente TFG, el autor posee y expone una serie de objetivos que se pueden agrupar de la siguiente forma:

- Objetivos formativos
- Objetivos de la aplicación
- Objetivos personales

### 1.2.1 Objetivos formativos

En este punto se pretende destacar el aporte formativo que brindará la realización del presente TFG al autor:

## 1.2 Objetivos

- Profundizar en el funcionamiento interno de Docker y en el uso de Docker-Compose
- Descubrir la forma en la que los lenguajes de programación integrados en la herramienta interactúan con el sistema operativo.
- Conocer de principio a fin el desarrollo y mantenimiento de una aplicación y extrapolarla a varios S.O.
- Aprender a utilizar la herramienta LaTeX, utilizada para la redacción del presente TFG.

### 1.2.2 Objetivos de la aplicación

El proceso de elaboración de UALTools está guiado por los siguientes objetivos a cumplir por la misma:

- Su funcionamiento debe estar basado en la ejecución de contenedores con Docker.
- Facilitar la instalación de UALTools en los sistemas operativos Windows 10, Linux y MacOS.
- Ejecución de programas simples o scripts en Java, Go y Python[4].
- Ejecución de servicios o aplicaciones de Java y Go que interactúen con bases de datos, y permitan la creación de APIs. De tal forma que se puedan crear entornos de desarrollo. Los servicios o aplicaciones del entorno se configurarán con un fichero de configuración llamado **ualtools.yml**, que la UALTools leerá para ejecutar todos los contenedores necesarios.
- Crear bases de datos SQL y NoSQL que interactúen con las aplicaciones dentro del entorno. En este caso MySQL y Redis.
- Lanzar migraciones para facilitar la puesta en marcha de la base de datos de MySQL y posibilitar la automatización de dicha funcionalidad con comandos.
- Dar la posibilidad de crear APIs Rest en cualquiera de los entornos que facilita UALTools.

## 1.3 Planificación

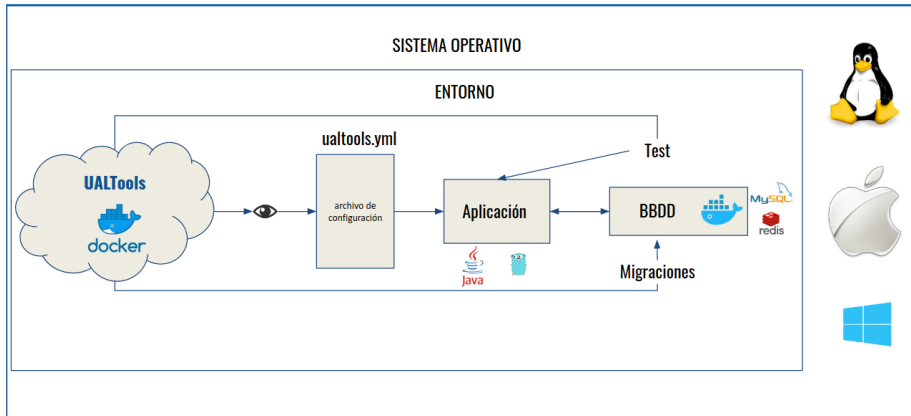


Figura 1: Representación de entornos de UALTools.

### 1.2.3 Objetivos personales

Obviando el fin de que UALTools y la presente documentación sean el medio para aprobar la asignatura de TFG, el autor tiene como objetivo ofrecer la herramienta a la docencia del Grado de Ingeniería Informática de la Universidad de Almería para si se ve conveniente, utilizarla en las asignaturas pertinentes.

## 1.3 Planificación

La planificación del proyecto se ha abordado en 3 fases principales, una de análisis del proyecto, otra de diseño del mismo, y por último otra fase de implementación que se ha llevado a cabo junto a las pruebas del mismo trabajo (ver Figura 2).

	Abril	Mayo	Junio	Julio	Agosto
Análisis de herramientas	2 semanas	1 semana			
Diseño UALTools		3 semanas	1 semana		
Implementación UALTools				2 semanas	2 semanas
Pruebas UALTools				3 semanas	1 semana
Redacción del TFG					1 semana

Figura 2: Planificación temporal.

Cada semana de planificación tiene un total de 25 horas estimadas



## 1.3 Planificación

### 1.3.1 Organización y seguimiento - Trello

Aquí una muestra del tablero de UALTools tras la finalización del proyecto de UALTools.

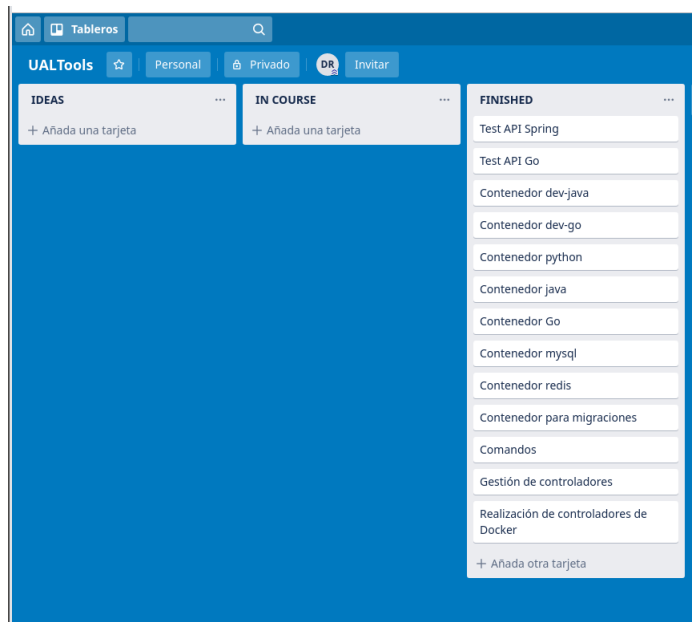


Figura 3: Panel Trello UALTools.

Como se puede ver en la Figura 3, todas las tareas están en una columna Finished. Durante el análisis del proyecto, primero se fueron añadiendo tarjetas a **IDEAS**, que durante la realización de dichas tareas, se pasaban a **IN COURSE** para terminar en la columna actual una vez se finalizaban dichas tareas.

## 1.4 GitHub

### 1.4 GitHub

Todo el progreso que ha ido subiéndose a la plataforma GitHub.

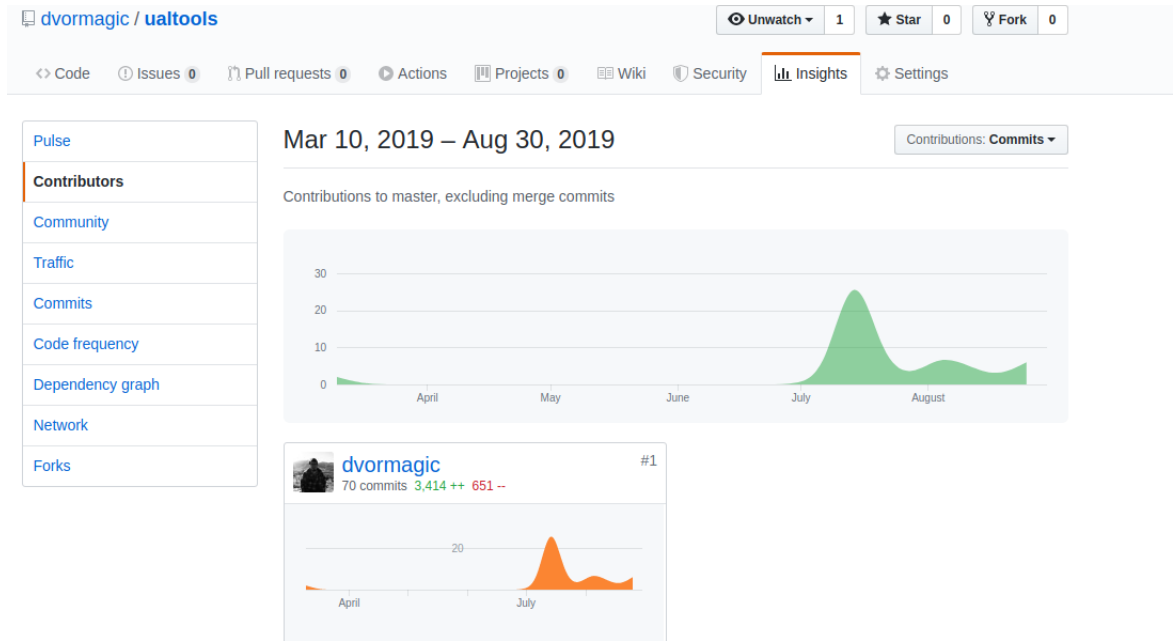


Figura 4: Progreso de commits en GitHub.

En la Figura 4 se muestra el volumen de commits del proyecto a lo largo de marzo/agosto de 2019.

A continuación, en las siguientes secciones se describen las principales tareas que han formado parte del desarrollo del proyecto.

#### 1.4.1 Análisis de herramientas

En esta fase, el objetivo principal era obtener el conocimiento necesario tanto de las herramientas que se iban a utilizar para la puesta en marcha del desarrollo y pruebas de UALTools (Docker, Docker-compose y Go) cómo de decidir e investigar el set de contenedores (en un contexto de herramientas) que iba a integrar UALTools (Go, Python, Docker, entornos en Go, entornos en Java, MySQL, Redis, phpmyadmin y una herramienta para lanzar migraciones). Dicho proceso duró unas 75 horas.

#### 1.4.2 Diseño UALTools

El Diseño de UALTools fue un proceso de aproximadamente unas 75 horas, 25 de ellas simultáneas con el punto de **Análisis de herramientas**, esta fase se

## 1.5 Estructura de la memoria

puede partir en 3 puntos:

- **Diseño de un sistema de Go que haga uso de Docker:** Tanto a nivel de comandos como a nivel de creación de entornos con distintos contenedores, ha sido necesario diseñar un sistema que permita extrapolar dichos mecanismos y así poder hacer uso de Docker a través de la aplicación.
- **Diseño de un entorno diferenciado de pruebas:** Dado que las imágenes de los contenedores de la aplicación se descargan desde Container Registry<sup>1</sup> de Google Cloud Platform[6], es importante poder testear cada una de las herramientas que se añaden a UALTools sin necesidad de subir las imágenes de los contenedores a la nube. Es por esto, que se ha añadido un fichero Docker compose para facilitar el testeo de nuevos contenedores a posteriori de subirlos a la plataforma.
- **Diseño de la estructura del fichero de configuración de entornos:** Para la creación de entornos, se decidió diseñar un fichero de configuración cuyo funcionamiento está inspirado en los docker-compose.yml<sup>2</sup>, de tal forma que UALTools es capaz de generar automáticamente el entorno haciendo una lectura de dicho documento, llamado **ualtools.yml**

### 1.4.3 Implementación UALTools & Pruebas UALTools

Estos dos pasos de la planificación, se han realizado de forma conjunta, dada la naturaleza de la aplicación, el autor consideró la necesidad de ir testeando la aplicación a la vez ésta iba a ir siendo implementada. La implementación se ha hecho en el orden y siguiendo los puntos especificados en el diseño.

### 1.4.4 Redacción del TFG

Se necesitó la última semana del mes de agosto de 2019, junto con ajustes puntuales en el desarrollo para redactar el documento, que está escrito en el lenguaje de programación L<sup>A</sup>T<sub>E</sub>X.

## 1.5 Estructura de la memoria

El TFG data de los siguientes capitulos:

1. **Introducción:** En este punto se expone una idea genérica de en qué consiste la aplicación UALTools, herramientas que se utilizan para el desarrollo y en qué consiste el proceso de elaboración del mismo
2. **Estado del arte:** Consiste en una enumeración de distintas herramientas/tecnologías que muestran funcionalidades similares a UALTools, exponiendo los puntos en común y distantes entre todo el conjunto expuesto en dicho punto.

---

<sup>1</sup>Aplicación de Google Cloud [2] dónde se guardan imágenes de contenedores, que posteriormente pasan a hacerse públicos.

<sup>2</sup>Fichero de configuración de docker-compose

## 1.5 Estructura de la memoria

3. **Tecnologías y herramientas:** Aquí se aborda el conjunto de tecnologías y herramientas que hacen posible la implementación y funcionamiento de UALTools.
  4. **UALTools:** Exposición detallada de cómo se ha desarrollado la aplicación, haciendo una descripción o explicación de cada uno de los procesos llevados a cabo.
  5. **Resultados:** Se expone el funcionamiento de la aplicación con casos reales y todas las posibilidades y experiencias que ofrece el uso de UALTools.
  6. **Conclusiones y trabajos futuros:** Este capítulo aborda las conclusiones del autor después de la realización del proyecto y de qué forma se podría mejorar la aplicación.
- **Bibliografía** Se enumeran las referencias y fuentes consultadas para la elaboración del presente TFG.

## 1.5 Estructura de la memoria

## 2 Estado del arte

Al igual que UALTools, existen varias herramientas que proporcionan al usuario la posibilidad de crear entornos de desarrollo, sin tener que invertir tiempo en buscar y configurar individualmente cada una de las herramientas.

Dos de las herramientas más conocidas que muestran ciertas similitudes con UALTools son XAMPP y Docker Compose:

### 2.1 XAMPP



Figura 5: Logo de XAMPP.

**XAMPP** es la herramienta para instalar en entornos fundamentalmente de desarrollo por excelencia. Dicha instalación, contiene un set básico de herramientas, entre las que se encuentran:

- **Apache:** Servidor HTTP (implementa protocolo http 1.1) de código abierto multiplataforma, fácil de usar y muy popular.



Figura 6: Logo de Apache.

- **mariadb:** Es un sistema de gestión de bases de datos basado en MySQL y desarrollado por la comunidad de desarrolladores de software libre y la fundación MariaDB. Presenta funcionalidades extra con respecto a MySQL (entre ellas más mecanismos de almacenamiento y una mayor facilidad de uso) convirtiéndolo en una alternativa más completa.

## 2.2 Docker compose



Figura 7: Logo de mariadb.

- **PHP:** Lenguaje de programación de 1995 que fue diseñado para el preprocesado de textos en formato UTF-8, pero que acabó siendo durante muchos años (aun sigue teniendo gran influencia) el lenguaje por excelencia para el lado del servidor en desarrollo web.



Figura 8: Logo de PHP.

- **Perl:** Lenguaje de programación de 1987, dicho lenguaje está inspirado en C, shell y AWK entre otros.



Figura 9: Logo de Perl.

## 2.2 Docker compose

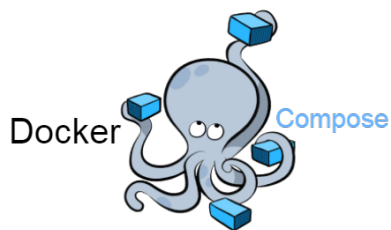


Figura 10: Logo de docker-compose.

**Docker compose** es un gestor de contenedores de Docker, que permite crear entornos con contenedores de haciendo uso de un fichero de configuración llamado **docker-compose.yml**, es decir, de una forma similar a cómo lo hace UALTools.

## 2.3 Comparativa

### 2.3 Comparativa

Una vez expuestas las herramientas más populares y que más se asemejan en funcionalidades a UALTools, sólo queda incidir en las diferencias principales, a grandes rasgos entre ellas.

	XAMP	Docker-compose	UALTools
Permite la creación de entornos de desarrollo			
Set de herramientas ampliable			(*)
Permite desarrollos en distintos lenguajes			
Enfocado a todo tipo de desarrollos			
No requiere conocimientos técnicos de otras herramientas			
No requiere conocimientos técnicos de la propia herramienta			
Requiere algún fichero de configuración			
Compatible con distintos sistemas operativos			
Proyecto público			

Figura 11: Tabla comparativa de herramientas similares.

En la Figura 11, se destaca que UALTools es un **Set de herramientas ampliable**, aunque no se pueda hacer con el uso de la misma aplicación (como sí se hace en el caso de docker-compose), se puede hacer de una forma sencilla y siempre es posible sugerir al autor un **Pull request**<sup>3</sup> a través del repositorio público del proyecto, ubicado en <https://github.com/dvormagic/ualtools> para añadir una herramienta que pueda ser útil para su uso.

---

<sup>3</sup>Acción de sugerir un cambio a un repositorio del que un usuario no es miembro, o desde otra rama del propio proyecto



### 2.3 Comparativa

## 3 Tecnologías y Herramientas

Este capítulo abordará todas las tecnologías que se han utilizado, una breve descripción de cada una de ellas, y el por qué es necesario utilizar dicha herramienta para el desarrollo de UALTools.

### 3.1 Ubuntu MATE

Ubuntu MATE [11] es una vertiente del sistema operativo Ubuntu<sup>4</sup> que utiliza el escritorio MATE. Haciéndolo bastante más ligero que la versión original, además, su navegabilidad se hace bastante más intuitiva y los menús son bastante personalizables. Además de todas las razones anteriores, el autor decidió usar este sistema operativo por la familiarización que tiene con él.



Figura 12: Logo de Ubuntu MATE.

---

<sup>4</sup>Distribución más usada de Linux

### 3.2 Postman

Postman es una herramienta pensada para el Testeo de APIs, que permite construir peticiones de una forma sencilla e intuitiva. Se ha utilizado para testear las APIs de los servicios de desarrollo de UALTools.



Figura 13: Logo de Postman.

### 3.3 Trello

Trello es la aplicación que se ha utilizado para llevar una organización de tareas y seguimiento del estado del proyecto. Ésta es una aplicación online que permite la gestión de tareas a través de un sistema de "tarjetas" (así se denominan dentro de la aplicación) que van pasando de un bloque a otro en el transcurso de ejecución del proyecto.



Figura 14: Logo de trello.

### 3.4 Go

#### 3.4 Go

Go [3] es un lenguaje de programación del año 2009 desarrollado por **Google**, es de tipo compilado, concurrente, imperativo y estructurado. De entre estas características, cabe destacar la de la concurrencia<sup>5</sup>, ya que como se verá en el siguiente apartado de UALTools, ha sido indispensable su uso para la puesta en marcha de la aplicación.



Figura 15: Logo de Go.

#### 3.5 Docker

Docker es un sistema de gestión de contenedores. Dichos contenedores se podrían denominar como unidades de software estándar, que empaquetan el código y todas sus dependencias para que dicha unidad o aplicación se pueda ejecutar de una forma rápida y confiable en distintos entornos y sistemas operativos.

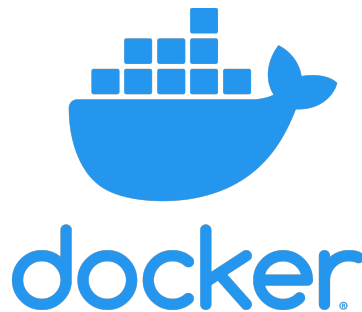


Figura 16: Logo de Docker.

##### 3.5.1 ¿Qué diferencia a los contenedores de una máquina virtual?

Principalmente, los contenedores realizan una virtualización del sistema operativo mientras que, una máquina virtual realiza una virtualización del hardware.

---

<sup>5</sup>Capacidad de lanzar procesos o hilos de ejecución creados por un mismo programa, de forma independiente y sin inferir entre ellas

### 3.5 Docker

¿Qué implica esto? Básicamente, para la ejecución de una máquina virtual, es necesario que el Hypervisor<sup>6</sup> del sistema operativo reserve los recursos de la propia máquina host para dedicarlos en mayor, o menor medida a las máquinas virtuales que estén funcionando en el sistema operativo.

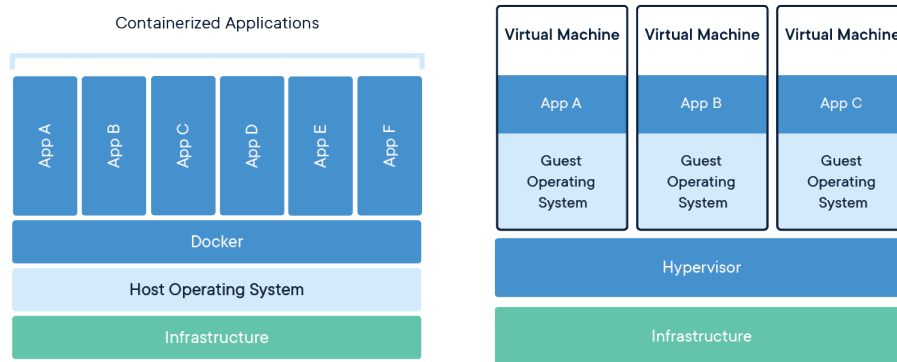


Figura 17: Virtualización de máquinas virtuales.

Por otra parte, los contenedores abstraen la capa de la aplicación del sistema operativo, permitiendo la agrupación del código y las dependencias. Se pueden ejecutar varios contenedores en un mismo Host, compartiendo éstos el kernel<sup>7</sup> del sistema operativo entre sí. Además, éstos se ejecutan como procesos aislados en el entorno del usuario.

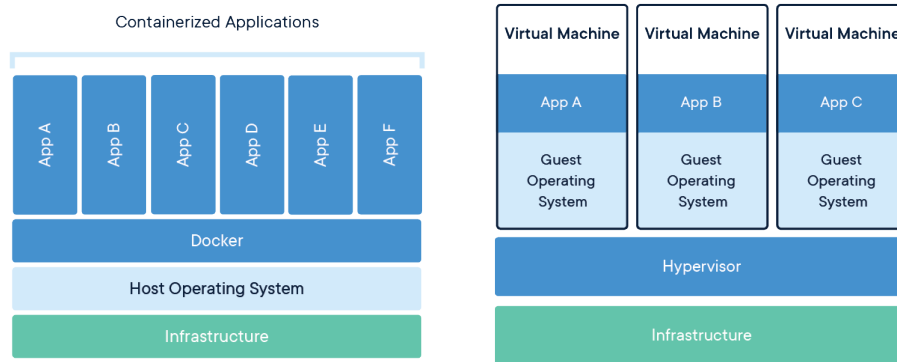


Figura 18: Virtualización de contenedores.

Una vez expuestas estas diferencias, es claro que los contenedores ocupan menos espacio que las máquinas virtuales, pueden controlar más aplicaciones y

<sup>6</sup>Plataforma encargada de la virtualización y monitorización de otros sistemas operativos dentro de un sistema operativo host.

<sup>7</sup>Núcleo de un sistema operativo

### 3.6 Docker compose

sobre todo, consumen menos recursos de la máquina host (ya que no es necesario que éstos sean reservados, si no que por así decirlo, éstos cogen los recursos que necesitan).

Finalmente, la decisión de usar Docker en UALTools viene dada además de por todos los puntos anteriores, por la facilidad de uso y abstracción que tiene. Dando la posibilidad de crear una interfaz en un lenguaje de programación (en este caso Go) que ejecute comandos por debajo y permita manejar esta herramienta de una forma programática, ajustándose a los requisitos necesarios de UALTools.

### 3.6 Docker compose

Como se ha mencionado anteriormente, y como se detallará más adelante, ha sido necesario el uso de ésta herramienta para el testeo de la aplicación. Docker compose ofrece la posibilidad de levantar un conjunto de contenedores con un sólo comando, y además permite sobrescribir un contenedor (o más bien, el tag del mismo) permitiendo así poder testear los contenedores la aplicación UALTools.

### 3.7 Sublime Text

Sublime Text<sup>[10]</sup> es un editor de texto creado para ser una extensión de vim que fue cogiendo vida propia con el paso del tiempo. Está escrito en C++ y utiliza Python para las extensiones. Se ha decidido usar esta herramienta por que es ligera, personalizable y además, el autor está muy familiarizado con ella.



Figura 19: Sublime Text logo.

### 3.8 Git

Software de control de versiones pensado para llevar un registro de los cambios en los archivos de un programa y además, para coordinar el trabajo de varias personas sobre un desarrollo conjunto.

El uso que se ha hecho de Git[7] en UALTools ha sido básicamente para ir guardando los cambios en un repositorio externo, a pesar de no haber trabajado con ramas, ya que el desarrollo ha sido llevado a cabo por una sola persona (el autor).



Figura 20: Git logo.

### 3.9 GitHub

GitHub es una plataforma de desarrollo colaborativo, dónde se alojan proyectos haciendo uso de Git. Por norma general, es utilizado para almacenar el código de programas o proyectos de software, ya sean públicos o privados.

UALTools aparece como un repositorio público en dicha plataforma (<https://github.com/dvormagic/ualtools>) dando la posibilidad a todos los usuarios de la misma de visualizar el código fuente, de sugerir cambios a través de pull requests y colaborar en el proyecto.



Figura 21: GitHub logo.

### 3.10 Cobra



Figura 22: Logo Cobra.

La librería Cobra [9] librería de Go permite tanto crear potentes aplicaciones modernas de CLI<sup>8</sup> como un programa para generar aplicaciones y archivos de comandos.

Muchos de los proyectos de Go más usados usan Cobra, entre ellos: **Kubernetes**, **Docker**, **OpenShift** o **CrocoachDB** entre otros muchos.

El uso que se le va a dar será el de definir comandos con funcionalidad propia de una forma muy sencilla.

### 3.11 GCloud



Figura 23: Google Cloud Platform logo.

Google Cloud Platform es la plataforma sobre la cual UALTools despliega sus contenedores y sus ficheros binarios para instalación. Para ello se hará uso de Container Registry y Google Cloud Storage respectivamente. Container Registry es la herramienta de Google Cloud Platform que permite gestionar contenedores de Docker en la nube y acceso a los mismos. Google Cloud Storage es el sistema de almacenamiento de Google Cloud Platform. Las instancias que

---

<sup>8</sup>Interfaz de línea de comandos



### 3.11 GCloud

almacenan información se hacen llamar Segmento (o Buckets). En el caso de UALTools, se usa un bucket llamado **ualtools**.

## 4 UALTools

Este punto es, posiblemente, el más importante del presente TFG. En él se abordará todo el proceso de desarrollo de UALTools.

### 4.1 Requisitos del sistema

#### 4.1.1 Requisitos generales

RG-001	Administrar herramientas
Dependencias	-
Descripción	El sistema debe permitir encender, apagar, reiniciar y eliminar cada una de las herramientas.
Comentarios	El usuario hará un uso sencillo y similar de todas las herramientas

RG-002	Servicios en Go
Dependencias	RG-001
Descripción	El sistema posibilitará la creación de un servicio/entorno de desarrollo en Go
Comentarios	No necesitará estar pendiente de configurar la herramienta.

RG-003	Servicios en Java
Dependencias	RG-001
Descripción	El sistema permitirá la creación de un servicio/entorno de desarrollo en Java
Comentarios	No necesitará estar pendiente de instalar y configurar la herramienta.

RG-004	Scripts en Go
Dependencias	RG-001
Descripción	El sistema permitirá la ejecución de Go con pequeños ficheros o scripts.
Comentarios	De igual forma que en los servicios, no será necesario configurar la herramienta.

#### 4.1 Requisitos del sistema

<b>RG-005</b>	<b>Scripts en Java</b>
<b>Dependencias</b>	RG-001
<b>Descripción</b>	Mismo caso de RG-004 aplicado a Java
<b>Comentarios</b>	-

<b>RG-006</b>	<b>Scripts en Python</b>
<b>Dependencias</b>	RG-001
<b>Descripción</b>	Mismo caso que RG-004 y RG-005 aplicado a Python
<b>Comentarios</b>	-

<b>RG-007</b>	<b>Arrancar MySQL</b>
<b>Dependencias</b>	RG-001
<b>Descripción</b>	El sistema permitirá levantar una Base de datos MySQL.
<b>Comentarios</b>	No será necesario configurarla ni instalarla.

<b>RG-008</b>	<b>Arrancar Redis</b>
<b>Dependencias</b>	RG-001
<b>Descripción</b>	Mismo caso que RG-007 aplicado a Redis.
<b>Comentarios</b>	-

<b>RG-009</b>	<b>Ejecutar Migraciones</b>
<b>Dependencias</b>	RG-001, RG-007
<b>Descripción</b>	El sistema permitirá lanzar migraciones para preparar bases de datos.
<b>Comentarios</b>	-

#### 4.1 Requisitos del sistema

<b>RG-010</b>	<b>Sincronización de herramientas</b>
<b>Dependencias</b>	RG-001, RG-002, RG-003, RG-004, RG-005, RG-006, RG-007, RG-008, RG-009, RG-010
<b>Descripción</b>	El sistema deberá permitir la sincronización entre herramientas.
<b>Comentarios</b>	Dicha sincronización permitirá la interacción entre herramientas.

##### 4.1.2 Requisitos no funcionales

<b>RNF-001</b>	<b>Ejecución</b>
<b>Descripción</b>	Será necesario tener instalado Docker.
<b>Comentarios</b>	Ya que el programa lanza comandos de Docker.

<b>RNF-002</b>	<b>Multiplataforma</b>
<b>Descripción</b>	El sistema debería ser compatible con al menos Linux y Windows.
<b>Comentarios</b>	Ya que por debajo ejecuta contenedores.

<b>RNF-003</b>	<b>Interacción</b>
<b>Descripción</b>	El sistema debe ser un programa de terminal.
<b>Comentarios</b>	Para habitar a los nuevos desarrolladores a usarlo.

<b>RNF-004</b>	<b>Depuración</b>
<b>Descripción</b>	El sistema debe mostrar los logs de cada herramienta.
<b>Comentarios</b>	Para así ofrecer una buena experiencia de depuración al usuario.

## 4.2 Diagrama de casos de uso

### 4.2 Diagrama de casos de uso

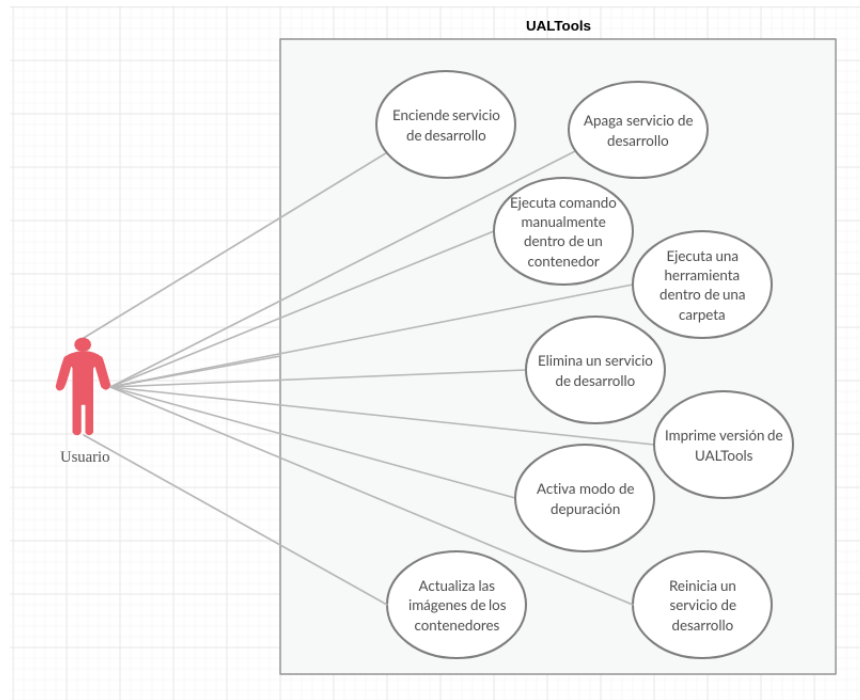


Figura 24: Diagrama casos de uso

La Figura 24 representa todos los casos de uso que se dan entre el usuario y la aplicación.

## 4.2 Diagrama de casos de uso

### 4.2.1 Especificación de actores del sistema

<b>A-001</b>	<b>Usuario</b>
<b>Dependencias</b>	-
<b>Descripción</b>	Usuario genérico de la aplicación.
<b>Comentarios</b>	Normalmente, será un desarrollador.

### 4.2.2 Especificación de casos de uso del sistema

<b>CU-001</b>	<b>Encender servicio de desarrollo</b>
<b>Dependencias</b>	RNF-001, RG-001, RG-002
<b>Precondición</b>	Se deberá haber configurado el fichero ualtools.yml
<b>Postcondición</b>	Se deberá arrancar el servicio deseado.
<b>Descripción</b>	El usuario usará el comando `ualtools start (herramienta)`
<b>Comentarios</b>	Podrá ser una aplicación de Go o Java. Sin argumentos los encenderá todos.

<b>CU-002</b>	<b>Apagar servicio de desarrollo</b>
<b>Dependencias</b>	RNF-001, RG-001, RG-002
<b>Precondición</b>	Deberá haber un servicio encendido.
<b>Postcondición</b>	El servicio dejará de funcionar.
<b>Descripción</b>	El usuario usará el comando `ualtools stop (herramienta)`
<b>Comentarios</b>	Sin argumentos la herramienta los parará todos.

<b>CU-003</b>	<b>Ejecutar comandos dentro de contenedores</b>
<b>Dependencias</b>	RNF-001, RG-XYZ (Excepto RG-007)
<b>Precondición</b>	Se deberá ejecutar dentro de un contenedor que exista en UALTools.
<b>Postcondición</b>	Aparecerá un terminal, dentro del contenedor dónde se podrán ejecutar comandos.
<b>Descripción</b>	El usuario ejecutará el comando `ualtools run (contenedor)`
<b>Comentarios</b>	-

## 4.2 Diagrama de casos de uso

<b>CU-004</b>	<b>Ejecutar herramienta dentro de una carpeta</b>
<b>Dependencias</b>	RNF-001, RG-004, RG-005, RG-006, RG-007, RG-008, RG-009
<b>Precondición</b>	El contenedor debe tener alguna herramienta asociada.
<b>Postcondición</b>	El contenedor deberá ejecutarse de forma que podría interaccionar con los ficheros de la carpeta.
<b>Descripción</b>	El usuario ejecutará el comando `ualtools (contenedor) (comando que se desee)`
<b>Comentarios</b>	Por ejemplo, para ejecutar un fichero en python: `ualtools python hello_world.py`

<b>CU-005</b>	<b>Elimina un servicio de desarrollo</b>
<b>Dependencias</b>	RNF-001, RG-001, RG-002
<b>Precondición</b>	El servicio de desarrollo debe haberse creado.
<b>Postcondición</b>	El servicio de desarrollo y su contenedor desaparecerán de la máquina host.
<b>Descripción</b>	El usuario ejecutará el comando `ualtools rm (servicio)`
<b>Comentarios</b>	-

<b>CU-006</b>	<b>Imprime versión de UALTools</b>
<b>Dependencias</b>	RNF-001
<b>Precondición</b>	-
<b>Postcondición</b>	Se mostrará la versión de UALTools
<b>Descripción</b>	El usuario ejecutará el comando `ualtools version`
<b>Comentarios</b>	-

<b>CU-007</b>	<b>Activa modo de depuración</b>
<b>Dependencias</b>	RNF-001, RG-004, RG-005, RG-006, RG-007, RG-008, RG-009
<b>Precondición</b>	-
<b>Postcondición</b>	Aparecerá información más detallada (comandos que ejecuta el contenedor) en la ejecución de herramientas.
<b>Descripción</b>	El usuario usará el comando `ualtools debug (herramienta) (comando)`
<b>Comentarios</b>	-

### 4.3 Arquitectura

<b>CU-008</b>	<b>Reinicia un servicio de desarrollo</b>
<b>Dependencias</b>	RNF-001, RG-001, RG-002
<b>Precondición</b>	El servicio deberá haberse encendido.
<b>Postcondición</b>	El servicio se apagará y volverá a encenderse
<b>Descripción</b>	El usuario ejecutará `ualtools restart (servicio)`
<b>Comentarios</b>	Sin argumento los reseteará todos.

<b>CU-009</b>	<b>Actualiza las imágenes de los contenedores</b>
<b>Dependencias</b>	RNF-001
<b>Precondición</b>	-
<b>Postcondición</b>	La aplicación descargará las imágenes actualizadas.
<b>Descripción</b>	El usuario ejecutará el comando `ualtools pull`
<b>Comentarios</b>	-

### 4.3 Arquitectura

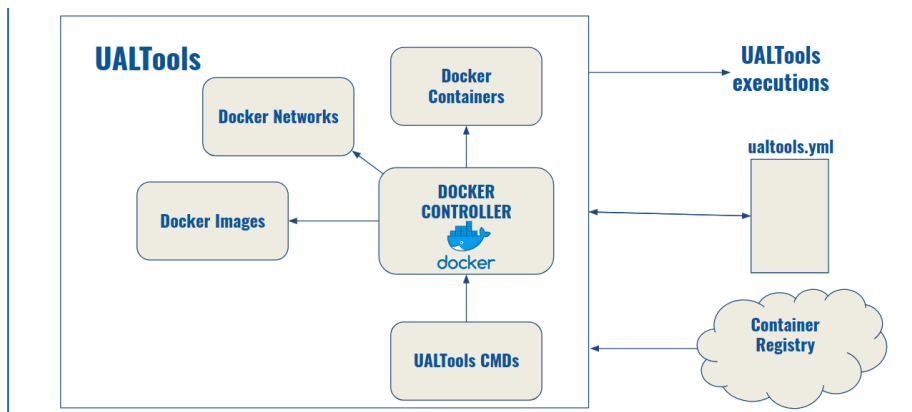


Figura 25: Arquitectura UALTools.

La arquitectura de UALTools se caracteriza, a grandes rasgos, por un punto de entrada de comandos, que de una forma u otra interactúan con varios controladores de Docker (en la Figura 25 se ha representado como DOCKER CONTROLLER). Estos controladores son:

- Controlador de contenedores: Será el encargado de ejecutar comandos de Docker sobre los contenedores de la aplicación.
- Controlador de imágenes: Será el encargado de descargar las imágenes de los contenedores y sobrescribirlas. Dichas imágenes se descargarán a través de Container Registry con el comando ‘ualtools pull’



## 4.4 Implementación

- **Controlador de redes:** Será el encargado de crear una red para cada contenedor, habrá varios tipos en función del tipo de contenedor o comando que se esté ejecutando.

Si se trata de comandos que ejecutan servicios o aplicaciones dentro de un entorno, UALTools usará el fichero `ualtools.yml` de dicho entorno para extraer la configuración de los contenedores y que el controlador trabaje con ellos.

## 4.4 Implementación

La estructura que del proyecto de UALTools es la siguiente:

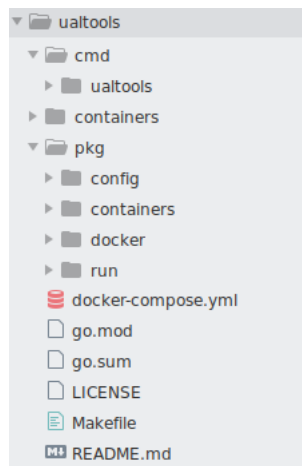


Figura 26: Árbol del proyecto UALTools.

En él se pueden apreciar 3 directorios principales, ésta sería una breve descripción de cada uno de ellos:

- **cmd:** Directorio entrypoint de la aplicación. Contiene todos los comandos que el usuario puede ejecutar para interactuar con la aplicación
- **containers:** Contiene todos los directorios con todas las herramientas que va a utilizar UALTools. Cada uno de estos directorios contiene un Dockerfile y algún fichero de bash en caso de que fuese necesario.
- **pkg:** Aquí está el core de la aplicación, donde está definida toda la funcionalidad de la misma. Dentro existen 4 subdirectorios que a grandes rasgos son:
  - **config:** Este directorio es en el que se añade todo el comportamiento de la aplicación a nivel de configuración. Tanto la parte de parsing del fichero `ualtools.yml` como un set de funciones para hacer ciertos tipos de comprobaciones durante el funcionamiento de la aplicación.

## 4.5 Controladores de Docker

- **containers:** Aquí es dónde se indica a UALTools cual es el set de herramientas que utiliza (o dicho de otra forma, cuales son los contenedores que va a utilizar), y qué características se desea que éstas tengan dentro del entorno o durante la ejecución de cada uno de ellos.
- **docker:** Este directorio es quizás el más relevante dentro de la aplicación. Es el encargado de dar presencia a los contenedores dentro de la aplicación. Es decir, contiene los controladores de Docker, dónde se definen la estructura de los contenedores, imágenes, redes y operaciones que se pueden realizar con ellos entre otro conjunto de operaciones necesarias para el correcto funcionamiento de UALTools.
- **run:** Este directorio es donde se establece la ejecución de comandos de docker, definidos a través de todos los controladores.

En los siguientes apartados, se explicará detalladamente cada uno de los puntos recientemente explicados, para entender de forma global el funcionamiento de UALTools.

## 4.5 Controladores de Docker

En este apartado se realizará un análisis exhaustivo de la forma en la que se ha abstraído la utilización de comandos de terminal a estructuras y funciones para utilizar todo lo que necesitamos en Go. Aquí se muestra el árbol de ficheros incluido en el directorio **pkg/docker**:

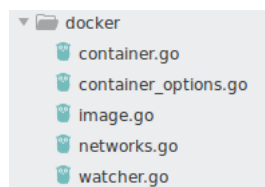


Figura 27: Árbol de ficheros pkg/docker.

### 4.5.1 pkg/docker/container.go

Este fichero contiene el esqueleto del gestor de contenedores de UALTools y toda su funcionalidad.

Aquí se muestra la estructura de dicho gestor:

## 4.5 Controladores de Docker

```
type ContainerManager struct {  
    name      string  
    image      *ImageManager  
    network    *NetworkManager  
    localUser  bool  
    networkAlias string  
    noTTY      bool  
    env        map[string]string  
    volumes    map[string]string  
    ports      map[int64]int64  
  
    // userWorkdir will overwrite workdir if specified  
    workdir    string  
    userWorkdir string  
  
    persistent bool  
}
```

Figura 28: Gestor de contenedores Docker.

Dicho gestor, posee una serie de campos que representan un conjunto de flags que se añaden a la hora de ejecutar los comandos ‘docker run ...’ o ‘docker create ...’ o ciertas propiedades de estos contenedores, en caso de que alguno de los campos tenga su representación como flag, se indicará en el punto correspondiente:

- **name:** Nombre que recibe el contenedor (o tag). El flag que representa es ‘-name’.
- **image:** Es la imagen desde la que se construye el contenedor, que está representada por un gestor de imágenes que se explicará más adelante.
- **network:** Al igual que la imagen, el contenedor también tendrá asociado un gestor de redes que será explicado más adelante. El flag que representa es ‘-network’.
- **localUser:** Este campo representa si el contenedor posee un usuario local, ya que hay ciertos casos, por ejemplo los contenedores de desarrollo/entorno de Go y de Java, en los que es necesario que el contenedor necesita ejecutar comandos con un usuario root. Por la propia naturaleza de los contenedores, no suelen traer usuarios configurados, a no ser que se añadan en el Dockerfile, como mostraremos más adelante para los dos casos mencionados. El flag que representa es ‘-user’.
- **networkAlias:** Alias asignado al contenedor en la red que está asociada al mismo. El flag que representa es ‘-network-alias’.
- **noTTY:** En este caso, se utiliza un campo para indicar si el contenedor posee una terminal a la que se pueda acceder para ejecutar comandos de forma interna. El flag que representa es ‘-t’.
- **env:** Este es el mapa para almacenar variables de entorno en el contenedor en caso de ser necesario en la construcción del mismo. El flag que representa es ‘-e’, es posible llamarlo varias veces en caso de que haya varias variables de entorno.

## 4.5 Controladores de Docker

- **volumes:** Este campo representa el mapa de volúmenes asociados del contenedor, es decir, directorios o ficheros externos que se añaden al contenedor durante la construcción. El flag que representa es ‘-v’, al igual que con env, se puede llamar varias veces en caso de haber varios.
- **ports:** Este campo contiene el mapa de puertos expuestos del contenedor. El flag que representa es ‘-p’, misma mecánica que con volumes y variables de entorno.
- **workdir:** Este campo representa el directorio de trabajo dónde estará ubicado el entorno del contenedor. El flag que representa es ‘-w’.
- **userWorkdir:** Este campo representa el directorio de trabajo de un usuario, en caso de ser especificado, sustituirá al campo workdir
- **persistent:** Representa si el contenedor es de tipo persistente o no. En el caso de los servicios que se especifican en el fichero ualtools.yml, serán de tipo persistente, ya que en caso de que se paren, interesa que se conserve el estado que tenían. El flag que representa es ‘-rm’, en caso de estar activo, el contenedor se eliminará una vez ejecutado. (Por ejemplo, a la hora de ejecutar scripts puntuales)

Seguidamente, aparece el constructor de contenedores:

```
func Container(name string, options ...ContainerOption) (*ContainerManager, error) {
    container := &ContainerManager{
        name:      fmt.Sprintf("%s %s", config.ProjectName(), name),
        noTTY:      config.CircleCI(),
        env:        make(map[string]string),
        volumes:    make(map[string]string),
        ports:     make(map[int64]int64),
    }

    for _, option := range options {
        if err := option(container); err != nil {
            return nil, errors.Trace(err)
        }
    }

    if container.userWorkdir != "" {
        container.workdir = container.userWorkdir
    }

    return container, nil
}
```

Figura 29: Constructor de contenedores.

Dónde básicamente, se genera un nuevo contenedor con los campos que anteriormente se han visto en caso de que se le indique a través de opciones por argumento.

Posteriormente, aparecen un conjunto de métodos que en casi todos los casos van a representar una serie de comandos de Docker que serán especificados en cada apartado cuando se ejecuten. Entre estos métodos cabe resaltar:

## 4.5 Controladores de Docker

- **Exist:** Comprueba si el contenedor indicado existe o no. Ejecuta por debajo `'docker inspect (container-name)'`.
- **Running:** Comprueba si el contenedor indicado se está ejecutando o no. Ejecuta por debajo `'docker inspect -f .State.Running (container-name)'`.
- **Stop:** Detiene la ejecución de un contenedor. Por debajo lanza el comando `'docker stop (container-name)'`.
- **Start:** Ejecuta el contenedor y lo crea en caso de no existir. Por debajo lanza el comando `'docker start (container-name)'`.
- **Kill:** Detiene y elimina el contenedor enviando al proceso en ejecución una señal SIGKILL. Ejecuta por debajo `'docker kill (container-name)'`.
- **Remove:** Detiene y elimina el contenedor. Ejecuta por debajo `'docker rm (container-name)'`.
- **Run:** Arranca el contenedor. Ejecuta por debajo `'docker run (flags)'`.
- **Create:** Crea un nuevo contenedor. Ejecuta por debajo `'docker create (flags)'`.

Como se ha mencionado antes del listado de métodos, **Run** y **Create** utilizan por debajo la función privada **buildCommand**, que se encarga de construir el comando del método que lo llame añadiendo todos los flags que necesiten ser activados, mencionados en la descripción del gestor de contenedores.

## 4.5 Controladores de Docker

### 4.5.2 pkg/docker/image.go

Este fichero contiene el gestor de imágenes de los contenedores de la aplicación.

```
package docker

import (
    "fmt"

    "github.com/juju/errors"
    "github.com/dvormagic/ualtools/pkg/run"
)

type ImageManager struct {
    name string
}

func Image(repo, name string) *ImageManager {
    return &ImageManager{
        name: fmt.Sprintf("%s/%s", repo, name),
    }
}

func (image *ImageManager) Pull() error {
    return errors.Trace(run.InteractiveWithOutput("docker", "pull", image.String()))
}

func (image *ImageManager) String() string {
    return fmt.Sprintf("%s:latest", image.name)
}
```

Figura 30: Constructor de imágenes.

A diferencia del gestor de contenedores, el gestor de imágenes es mucho más simple. La estructura que representa a las imágenes, **ImageManager** sólo contiene un campo **name**. Por otra parte, el constructor sólo permite enviarle 2 strings, uno llamado **repo** (entiendase como el id o directorio de container registry dónde se almacenan las imágenes) y otro llamado **name**, que retorna un **ImageManager** con un nombre.

Por último, este fichero contiene el método **Pull**. Éste método ejecuta el comando ‘docker pull (image-name)’ para descargar la última versión de la imagen almacenada en container registry.

### 4.5.3 pkg/docker/network.go

Este fichero contiene el gestor de redes de los contenedores de la aplicación.

## 4.5 Controladores de Docker

```
package docker

import (
    "os/exec"

    "github.com/juju/errors"
    log "github.com/sirupsen/logrus"
)

type NetworkManager struct {
    name string
}

func Network(name string) *NetworkManager {
    return NetworkWithRealname("ualtools_" + name)
}

func NetworkWithRealname(name string) *NetworkManager {
    return &NetworkManager{name: name}
}

func (network *NetworkManager) Exists() (bool, error) {
    cmd := exec.Command("docker", "network", "inspect", network.name)
    if err := cmd.Run(); err != nil {
        if !cmd.ProcessState.Success() {
            return false, nil
        }
        return false, errors.Trace(err)
    }
    return true, nil
}

func (network *NetworkManager) Create() error {
    log.WithFields(log.Fields{"network": network.name}).Info("Create network")

    cmd := exec.Command("docker", "network", "create", network.name)
    return errors.Trace(cmd.Run())
}

func (network *NetworkManager) CreateIfNotExists() error {
    exists, err := network.Exists()
    if err != nil {
        return errors.Trace(err)
    }

    if !exists {
        return errors.Trace(network.Create())
    }

    return nil
}

func (network *NetworkManager) String() string {
    return network.name
}
```

Figura 31: Constructor de redes.

Este controlador se asemeja bastante al de imágenes por el hecho de que solamente hay un campo **name** en el **NetworkManager**. Pero en este caso, el constructor creará una red con el nombre ‘ualtools\_(network-name)’. De tal forma que todas las redes de todos los contenedores estarán dentro de un mismo contexto.

Entre los métodos de este controlador, se van a destacar los 3 más importantes.

- **Exists**: Este método ejecuta por debajo el comando ‘docker network inspect (network-name)’, que básicamente realiza una comprobación de si la red existe o no.

## 4.5 Controladores de Docker

- **Create**: Este método ejecuta el comando ‘docker network create (network-name)’, que se encarga de crear una nueva red con el nombre que posee el controlador.
- **CreateIfNotExist**: Este método ejecuta el método anteriormente mencionado, solamente en el caso de que la red no exista, haciendo uso también del otro método de este fichero **Exists**. Este método se llama en el método privado anteriormente mencionado del fichero **container.go**, **buildCommand**, que decíamos que se encargaba de completar los comandos de **Run** y **Create**, para ser posteriormente ejecutados.

### 4.5.4 pkg/docker/container\_options.go

Este fichero es dónde están las funciones que son llamadas en el constructor para añadir los flags que nos interesan. El conjunto de funciones de este fichero están implementadas de una forma peculiar, proveniente de la programación funcional. A este tipo de funciones se les llama **Opciones funcionales**, porque básicamente son eso. Funciones que se ejecutan opcionalmente (solamente en caso de que hayan sido llamadas como argumento) y se ejecutan dentro del constructor en este caso.

Para entender el mecanismo de este tipo de funciones, se procede a explicar un caso concreto:

```
func WithImage(image *ImageManager) ContainerOption {  
    return func(container *ContainerManager) error {  
        container.image = image  
        return nil  
    }  
}
```

Figura 32: Opción funcional WithImage.

Ahora, recapitulando al constructor del gestor de contenedores, como segundo argumento tenía un array de **options** que eran de tipo **ContainerOption**, que a su vez es el tipo que devuelve **WithImage** (al igual que el resto de opciones funcionales de este fichero). ¿Qué es ContainerOption?

```
type ContainerOption func(container *ContainerManager) error
```

Figura 33: Tipo ContainerOption.

Es una función. Por tanto, al constructor del gestor de contenedores se le está pasando un array de funciones, que retornan funciones. Posteriormente, para ejecutar dichas funciones, el gestor hace esto por dentro:



## 4.5 Controladores de Docker

```
container := &ContainerManager{
    name:      fmt.Sprintf("%s %s", config.ProjectName(), name),
    noTTY:     config.CircleCI(),
    env:       make(map[string]string),
    volumes:   make(map[string]string),
    ports:     make(map[int64]int64),
}

for _, option := range options {
    if err := option(container); err != nil {
        return nil, errors.Trace(err)
    }
}
```

Figura 34: Ejecución de opciones funcionales.

En primer lugar, se crea el **ContainerManager** vacío, y a través de las opciones funcionales, añadimos aquellos parámetros que queramos. De tal forma, que si por ejemplo hubiesemos llamado al constructor y se le hubiese enviado **WithImage**, se asignaría la imagen indicada al gestor del contenedor.

Además del método ejemplificado, este fichero contiene las siguientes funciones:

- **WithNetwork**: Añade una red que se le especifique al contenedor en cuestión.
- **WithDefaultNetwork**: Añade una red con un nombre por defecto, haciendo uso del nombre del directorio en que se encuentra ubicado el usuario que ejecuta el contenedor en cuestión del formato ‘ualtools\_(dirname)’.
- **WithLocalUser**: Establece el campo del gestor de contenedores **localUser** a true.
- **WithSharedSSHSocket**: Transfiere el socket ssh de la máquina host al contenedor.
- **WithNetworkAlias**: Añade un network alias al gestor de contenedores.
- **WithoutTTY**: Establece a true el campo del gestor de contenedores **noTTY**.
- **WithSharedWorkspace**: Establece un workspace común entre todos los contenedores que tengan habilitada esta opción y habilita un directorio **workspace** como workdir dentro del contenedor.
- **WithEnv**: Añade una variable de entorno al contenedor. Esta función se puede enviar varias veces al constructor.
- **WithVolume**: Añade un volumen compartido desde el exterior al gestor. Puede ser llamada varias veces en un mismo constructor.
- **WithPort**: Expone un puerto del contenedor a la máquina host. Al igual que las dos anteriores, puede ser llamado en varias ocasiones.
- **WithSharedGopath**: Por la forma en la que Go gestiona los paquetes, es necesario añadir una serie de volúmenes específicos a través de esta función al constructor del gestor del contenedor.

## 4.5 Controladores de Docker

- **WithWorkdir**: Establece el **workdir** que contiene el contenedor.
- **WithPersistence**: Establece a true el campo **persistent** del contenedor.
- **WithStandardHome**: Establece la variable de entorno **\$HOME** como **"home/container"**

### 4.5.5 pkg/docker/watcher.go

En este fichero, se establece un elemento necesario para la ejecución de entornos en UALTools. La lógica que hay detrás de la ejecución de entornos es la de 1º poder lanzar en procesos independientes cada uno de los contenedores, 2º que estos se vayan arrancando de forma ordenada y 3º tener un logging personalizado de la actividad de los mismos.

Es aquí donde aparece la necesidad de crear la estructura **Watcher**:

```
type Watcher struct {
    wg      *sync.WaitGroup
    notifyExit chan struct{}
}

func NewWatcher() *Watcher {
    return &Watcher{
        wg:      new(sync.WaitGroup),
        notifyExit: make(chan struct{}),
    }
}
```

Figura 35: Definición y constructor de Watcher.

Esta es una de las razones por las cuales se ha sacado provecho de utilizar Go, y es la de utilizar **goroutines**<sup>9</sup>. Observando la estructura de **Watcher** se distinguen dos campos.

- **wg**: Abreviatura de WaitGroup, es un tipo de estructura encontrado en el paquete nativo **sync** de Go. Se puede entender como un controlador de goroutines. Hay que indicarle un número de goroutines que pueden ejecutarse simultáneamente (método **wc.Add(1)** en este caso), y hacer que las que terminen de ejecutarse envíen una señal de que han terminado (**wc.Done()**), una vez llega la señal, se lanzará otra goroutine desde una colección con otras goroutines que se encuentran esperando (gracias al comando **wc.Wait()**).

---

<sup>9</sup>Característica de Go que permite lanzar varios hilos de forma simultánea en un mismo proceso, optimizando los tiempos de procesamiento y dándole el atributo de concurrente a este lenguaje de programación

## 4.5 Controladores de Docker

```
func (watcher *Watcher) Run(serviceName string, container *ContainerManager) {
    watcher.wg.Add(1)
    go runForeground(watcher.wg, watcher.notifyExit, serviceName, container)
}

func (watcher *Watcher) Wait() {
    go func() {
        c := make(chan os.Signal)
        signal.Notify(c, os.Interrupt)
        for range c {
            // Newline to always jump the next log we emit.
            fmt.Println()

            // Enter a loop until all the containers exits and the main closes the whole app directly.
            for {
                watcher.notifyExit <- struct{}{}
            }
        }
    }()

    // Wait for all running services to finish.
    watcher.wg.Wait()
}
```

Figura 36: Métodos Run y Wait del watcher.

Aquí se puede observar cómo, al llamar a **Run()** se indica al WaitGroup que no permita ejecutar más de una goroutine simultánea. Posteriormente, se define el método **Wait()** dónde se indicará al WaitGroup que espere para ejecutar otro hilo hasta que reciba una señal de **Done()**.

## 4.5 Controladores de Docker

```
func runForeground(wg *sync.WaitGroup, notifyExit chan struct{}, serviceName string, container *ContainerManager) {
    defer wg.Done()

    logger := log.WithField("service", serviceName)
    logger.Info("Start service")

    notifyErr := make(chan error, 1)
    go func() {
        if err := container.Create(); err != nil {
            notifyErr <- err
            return
        }

        cmd := exec.Command("docker", "start", "-a", container.String())
        cmd.Stdin = os.Stdin

        loggerOut := log.New()
        loggerOut.Formatter = newPrefixFormatter(serviceName)
        loggerOut.SetLevel(log.StandardLogger().Level)
        cmd.Stdout = &logrusWriter{logger: loggerOut}

        loggerErr := log.New()
        loggerErr.Formatter = newPrefixFormatter(serviceName)
        loggerErr.SetLevel(log.StandardLogger().Level)
        cmd.Stderr = &logrusWriter{logger: loggerErr, debug: true}

        if err := cmd.Start(); err != nil {
            notifyErr <- err
            return
        }
        if err := cmd.Wait(); err != nil {
            if err.Error() == "signal: interrupt" {
                return
            }
        }

        notifyErr <- err
        return
    }()

    select {
    case err := <-notifyErr:
        logger.WithField("err", err.Error()).Error("Service failed")

    case <-notifyExit:
        logger.Info("Kill service")
        if err := container.Kill(); err != nil {
            logger.WithField("err", err.Error()).Error("Stop service failed")
        }
    }
}
```

Figura 37: Método privado runForeground.

Por último, el método privado **runForeground**, que es el que se encarga de crear/arrancar los contenedores en una goroutine, hace una llamada al principio del mismo a **wg.Done()** con un **defer**<sup>10</sup>

- **notifyExit**: Este campo representa otro aspecto muy interesante de Go, que es el de los **canales**.<sup>11</sup>.

Haciendo referencia de nuevo a las Figuras 36 y 37. En la primera, se puede apreciar cómo en el método **Wait()** antes de lanzar el **Wait()** del **wg**, se lanza una goroutine que básicamente, captura señales de **Ctrl+C**<sup>12</sup>.

<sup>10</sup>Etiqueta del Go que permite indicar a una función u operación que se ejecute al final de un proceso, en este caso, al final de la ejecución del método Run

<sup>11</sup>Los Canales son las tuberías que conectan goroutines concurrentes. Se pueden enviar valores por un canal de una goroutine y recibir esos valores en otra goroutine.

<sup>12</sup>Señal de interrupción

## 4.5 Controladores de Docker

En caso de detectarla, se enviará esta señal a todos los posibles canales abiertos del wg, indicándole que los procesos deben terminar.

En la Figura 37, se puede apreciar cómo al final aparece un selector, que hace una operación (haciendo uso de otro canal) en caso de que haya habido un error en la ejecución del contenedor en el propio método privado **runForeground**, o por otro lado devuelve otro error en caso de que la señal haya sido interrumpida por el usuario.

Por último, hacer mención al tercer punto que se comentaba en la introducción de este apartado. El logging personalizado de la actividad que registra cada una de las herramientas o servicios en ejecución. Para ello se ha utilizado la librería logrus [8] <https://github.com/sirupsen/logrus>. Que ofrece una gran personalización de los logs, aportando más claridad cuando se trata de buscar errores o identificar fallos en la terminal.

```
type prefixFormatter struct {
    t      *log.TextFormatter
    serviceName string
}

func (f *prefixFormatter) Format(entry *log.Entry) ([]byte, error) {
    entry.Message = fmt.Sprintf("(%s) %s", f.serviceName, entry.Message)
    return f.t.Format(entry)
}

func newPrefixFormatter(serviceName string) *prefixFormatter {
    return &prefixFormatter{
        t:      new(log.TextFormatter),
        serviceName: serviceName,
    }
}

type logrusWriter struct {
    logger *log.Logger
    debug bool
}

func (w *logrusWriter) Write(b []byte) (int, error) {
    const maxLogSize = 64 * 1024

    var result [][]byte
    parts := bytes.Split(bytes.TrimSpace(b), []byte("\r\n"))
    for _, part := range parts {
        for len(part) > maxLogSize {
            result = append(result, part[:maxLogSize])
            part = part[maxLogSize:]
        }

        if len(part) > 0 {
            result = append(result, part)
        }
    }
    for _, line := range result {
        if w.debug {
            w.logger.Debug(string(line))
        } else {
            w.logger.Info(string(line))
        }
    }
    return len(b), nil
}
```

Figura 38: Personalización y logging de las herramientas y servicios.

En primer lugar, se crea una estructura privada **prefixFormatter**. En

## 4.6 Especificación de contenedores

la librería utilizada para los logs, un formateador es una interfaz que tiene que devolver una función **func (f \*formater) Format(entry \*log.Entry) ([]byte, error)**. Básicamente, el formateador que se pretende crear es uno que imprima el nombre del servicio y el mensaje de entrada en cada log.

En segundo lugar, se crea un writer, que cumple con el requisito de interfaz impuesto para los writer en el paquete **os/exec** de Go (que posea un método **Write** como el que se ha implementado), cuando queremos pintar la salida de un comando. Este writer limitará el tamaño de los logs, para evitar imprimir logs excesivamente grandes y permitirá activar el login de depuración en caso de que se le indique.

Se puede observar el uso del formateador y del writer en la Figura 37, dentro de la goroutine, justo después de crear y arrancar el contenedor que esté procesándose en ese momento.

## 4.6 Especificación de contenedores

Una vez explicados los controladores de Docker creados en Go. El siguiente paso importante es el de echar un vistazo al conjunto de herramientas (contenedores) que van a ser utilizados por UALTools, y de qué forma se han configurado. Todos ellos están ubicados en el directorio **pkg/containers**.

Antes de nada, se ha creado una estructura auxiliar a **ContainerManager** que contiene la configuración previa que se le va a aplicar a cada ContainerManager en el futuro.

```
package containers

import (
    "github.com/juju/errors"
    "github.com/dvormagic/ualtools/pkg/docker"
)

const Repo = "eu.grc.io/ual-tools"

type Container struct {
    Image string
    Tools []string
    Options []docker.ContainerOption
}

var containers = []Container{
{
    Image: "dev-go",
    Tools: []string{},
    Options: []docker.ContainerOption{
        docker.WithSharedWorkspace(),
        docker.WithLocalUser(),
        docker.WithSharedGopath(),
        docker.WithStandardHome(),
    },
},
{
    Image: "dev-java",
    Tools: []string{},
    Options: []docker.ContainerOption{
        docker.WithSharedWorkspace(),
        docker.WithLocalUser(),
        docker.WithStandardHome(),
    },
},
}
```

Figura 39: Comienzo del código fuente de **pkg/containers/containers.go**

Se inicializa una constante **const Repo = "eu.grc.io/ual-tools"**, que

## 4.6 Especificación de contenedores

es básicamente la ruta dónde se encuentran las imágenes de UALTools en producción.

Después aparece una estructura Container. Esta es la estructura auxiliar mencionada anteriormente. Sólo tiene 3 campos:

- **Image:** Aquí va el nombre de la imagen. Posteriormente se explicará cómo se hace uso de la constante inicializada anteriormente y del nombre de la imagen para descargar la imagen del contenedor.
- **Tools:** Aquí se especifican las herramientas que hacen uso de dicha imagen, estas herramientas sólo se utilizarán para casos en los que no sea necesario tener un entorno preparado.
- **Options:** Se especifican las opciones que se mencionaban en el punto 4.5.4

Posteriormente, se inicializa un slice (array) de la estructura que se acaba de definir con todos los contenedores de la aplicación. En la figura se aprecia **dev-go**, que es el contenedor pensado para el servicio/entorno de desarrollo, por eso no tiene el campo Tools vacío y tiene las opciones (todas ellas explicadas en el punto 4.5.4):

- WithSharedWorkspace
- WithLocalUser
- WithSharedGopath
- WithStandardHome

También aparece reflejado la imagen **dev-java**, que posee unas características similares.

Además de estos dos contenedores. El slice presenta los siguientes:

- **go:** Para ejecución de scripts de Go. Las herramientas **go** y **gofmt** hacen uso de esta imagen y habilita las opciones: **WithSharedWorkspace WithLocalUser WithSharedGopath WithStandardHome WithSharedSSHSocket**
- **java:** Para ejecución de scripts de Java. La herramienta **java** hace uso de esta imagen para su ejecución y utiliza las opciones: **WithSharedWorkspace, WithLocalUser, WithStandardHome y WithSharedSSHSocket.**
- **python:** Para ejecución de scripts de Python. La herramienta **python** hace uso de esta imagen y habilita las mismas opciones que el contenedor de Java.
- **mysql:** Para habilitar una base de datos MySQL. La herramienta **mysql** hace uso de esta imagen y habilita las opciones: **WithSharedWorkspace y WithoutTTY**

## 4.7 Ejecución de contenedores

- **phpmyadmin**: Panel de consulta de la base de datos MySQL. No es usado por ninguna herramienta ni habilita ningún tipo de opción. Esta herramienta simplemente se expone en un puerto en la configuración y se habilita en una ruta local.
- **migrator**: Herramienta utilizada para realizar las migraciones de la base de datos MySQL. Las herramientas **migrator** e **init-migrator** hacen uso de este contenedor. Habilita las opciones **WithSharedWorkspace** y **WithStandardHome**
- **redis**: Habilita un servicio de Redis. La herramienta **redis-cli** hace uso de esta imagen y habilita la opción **WithoutTTY**

Por último, este fichero tiene 3 funciones:

- **Images()**: Retorna un listado de las imágenes de cada uno de los contenedores mencionados anteriormente.
- **List()**: Retorna el listado completo de contenedores.
- **FindImage()**: Se le envía una imagen por argumento y devuelve su contenedor correspondiente o un error de no encontrado.

## 4.7 Ejecución de contenedores

Haciendo referencia al punto **4.5.1**, dónde se aborda el tema de el controlador de contenedores. Se menciona el hecho de cada uno de los métodos se encarga de montar un comando de terminal usando Docker (ya sea directamente en el propio método o haciendo uso por debajo de la función auxiliar **buildCommand**), pues una vez construido el comando, todos los métodos ejecutan una de las funciones ubicadas en el fichero **pkg/run/interactive.go**. Las funciones son las siguientes:

- **Interactive**:



## 4.7 Ejecución de contenedores

```
func Interactive(name string, args ...string) error {
    cmd := exec.Command(name, args...)
    cmd.Stdin = os.Stdin

    loggerOut := log.New()
    loggerOut.SetLevel(log.StandardLogger().Level)
    wout := loggerOut.WriterLevel(log.DebugLevel)
    defer wout.Close()
    cmd.Stdout = wout

    loggerErr := log.New()
    loggerErr.SetLevel(log.StandardLogger().Level)
    werr := loggerErr.WriterLevel(log.ErrorLevel)
    defer werr.Close()
    cmd.Stderr = werr

    // Print the command. Do not use fields as they escape the value and the
    // result cannot be copied.
    logArgs := []string{}
    for _, arg := range args {
        if strings.Contains(arg, `"`) {
            logArgs = append(logArgs, strconv.Quote(arg))
        } else {
            logArgs = append(logArgs, arg)
        }
    }
    shell := fmt.Sprintf("%s %s", name, strings.Join(logArgs, " "))
    log.Debug("Run interactive command")
    log.Debugln(shell)

    if err := cmd.Run(); err != nil {
        return errors.Trace(err)
    }

    return nil
}
```

Figura 40: Función de ejecución de comandos Interactive.

Esta función se encarga básicamente de ejecutar dichos comandos y añadir a los logs de debug y de errores una etiqueta de debug o de error, además, no mostrará los logs de tipo Info. Se utiliza en el método **Run()** del controlador de contenedores.

## 4.8 Configuración de la aplicación

- InteractiveWithOutput

```
func InteractiveWithOutput(name string, args ...string) error {
    cmd := exec.Command(name, args...)
    cmd.Stdin = os.Stdin
    cmd.Stdout = os.Stdout
    cmd.Stderr = os.Stderr

    // Print the command. Do not use fields as they escape the value and the
    // result cannot be copied.
    logArgs := []string{}
    for _, arg := range args {
        if strings.Contains(arg, `"`) {
            logArgs = append(logArgs, strconv.Quote(arg))
        } else {
            logArgs = append(logArgs, arg)
        }
    }
    shell := fmt.Sprintf("%s %s", name, strings.Join(logArgs, " "))
    log.Debug("Run interactive command with output")
    log.Debugln(shell)

    if err := cmd.Run(); err != nil {
        return errors.Trace(err)
    }

    return nil
}
```

Figura 41: Función de ejecución de comandos InteractiveWithOutput.

Esta función muestra directamente los logs de salida del contenedor en la máquina Host. Sin tratar dichos logs de ninguna forma.

## 4.8 Configuración de la aplicación

En este apartado se tratará de explicar todo lo referente a **pkg/config**, que es dónde se encuentra todo lo referente a la configuración de UALTools. Se abordará este apartado haciendo mención a cada uno de los ficheros del mismo.

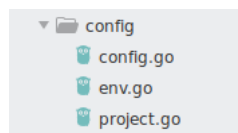


Figura 42: Árbol de directorio de configuración.

### 4.8.1 pkg/config/config.go

Este fichero es el que se encarga de leer el fichero de configuración **ualtools.yml** y lo parsea a estructuras de Go para posteriormente sacar la preparación de los entornos de desarrollo que se configuren.

## 4.8 Configuración de la aplicación

```
package config

import (
    "io/ioutil"
    "os"

    log "github.com/sirupsen/logrus"
    "gopkg.in/yaml.v2"
)

var Settings = new(Config)

func init() {
    content, err := ioutil.ReadFile("ualtools.yaml")
    if err != nil {
        if os.IsNotExist(err) {
            return
        }
        log.Fatal(err)
    }

    if err := yaml.Unmarshal(content, &Settings); err != nil {
        log.Fatal(err)
    }
}

type Config struct {
    Project string `yaml:"project"`

    Services map[string]*Service `yaml:"services"`
    Tools    map[string]*Tool    `yaml:"tools"`
}
```

Figura 43: Inicio del fichero pkg/config/config.go.

En primer lugar, aparece una variable global **Settings** que es de tipo **Config**. Dentro tiene 3 campos:

- **Project**: Este campo se encarga de definir el nombre del proyecto, dentro del mismo se podrán definir distintos servicios (unos en Go, otros en Java) que se comuniquen entre sí sin problema.

```
func (cnf *Config) IsService(name string) bool {
    _, ok := cnf.Services[name]
    return ok
}

func (cnf *Config) IsTool(name string) bool {
    _, ok := cnf.Tools[name]
    return ok
}

type Service struct {
    Type      string      `yaml:"type"`
    Deps      []string    `yaml:"deps"`
    Ports     []string    `yaml:"ports"`
    Workdir   string      `yaml:"workdir"`
    Volumes   []string    `yaml:"volumes"`
    Env       map[string]string `yaml:"env"`
}

type Tool struct {
    Container string `yaml:"container"`
    Deps      []string `yaml:"deps"`
    Ports     []string `yaml:"ports"`
    Volumes   []string `yaml:"volumes"`
}
```

Figura 44: Servicios y herramientas de la configuración.

- **Services**: Es la parte dónde se definen los servicios o "aplicaciones" que queramos ejecutar en nuestro entorno. La estructura sobre la que se monta cada servicio tiene los campos:

## 4.8 Configuración de la aplicación

- **Type:** Decide el tipo de servicio que va a representar, si es Go o Java.
  - **Deps:** Aquí es dónde se le indica qué dependencias necesita para funcionar correctamente. Por ejemplo, si un servicio interactúa con otro servicio, o un servicio interactúa con una de las herramientas, será necesario mencionar ese servicio o herramienta dentro de este campo.
  - **Ports:** Es dónde se indican los puertos que van a exponerse.
  - **Workdir:** Este campo contiene el workdir, en caso de que interese indicarlo.
  - **Volumes:** En este campo, se indican los volúmenes que se van a compartir con la máquina host.
  - **Env:** Aquí se indican las variables de entorno con las que se pretende que el servicio arranque.
- **Tools:** Define las herramientas o dependencias que será necesario habilitar para el correcto funcionamiento de alguno de los servicios habilitados anteriormente. Para este caso, **MySQL**, **Redis** o **phpmyadmin** son los 3 tipos de herramientas que se pueden definir. La estructura que contiene a dichas herramientas tiene los campos:
    - **Container:** Define el contenedor del que va a hacer uso esta herramienta, por ejemplo. Podría definirse una herramienta "database" que usase el contenedor de MySQL. A los ojos del resto de servicios, la herramienta será database.
    - **Deps:** De la misma forma que en los servicios, aquí se deben introducir las dependencias de la herramienta.
    - **Ports:** Mismo mecanismo que en los servicios, en este caso los puertos que se exponen.
    - **Volumes:** Similar a los volúmenes de las herramientas.

### 4.8.2 pkg/config/env.go

Este fichero contiene funciones que realizan comprobaciones sobre si hay ciertas variables de configuración habilitadas.

## 4.8 Configuración de la aplicación

```
package config

import (
    "os"
    "runtime"
)

func SSHAgentSocket() string {
    return os.Getenv("SSH_AUTH_SOCK")
}

func Windows() bool {
    return runtime.GOOS == "windows"
}

func Home() string {
    if Windows() {
        return os.Getenv("HOMEPATH")
    }
    return os.Getenv("HOME")
}
```

Figura 45: Fichero de comprobación de variables de entorno.

Este fichero posee 4 funciones:

- **SSHAgentSocket**: Retorna la variable de entorno **SSH\_AUTH\_SOCK**.
- **Windows**: Retorna **true** o **false** en función de si el sistema operativo en el que se está ejecutando es Windows o no.
- **Home**: Retorna la variable de entorno **HOMEPATH** en Windows y **HOME** en otros sistemas operativos.

### 4.8.3 pkg/config/project.go

Este fichero contiene una función privada `init`, que inicializa dos variables globales. Una que establecerá el nombre del proyecto y otra que establecerá el paquete del proyecto en función del **Project** que se indique en el fichero **ualtools.yml**. Además, se pueden apreciar dos funciones públicas que devuelven el valor de las dos variables mencionadas anteriormente.

## 4.9 Contenedores base de UALTools

```
package config

import (
    "os"
    "path/filepath"

    log "github.com/sirupsen/logrus"
)

var (
    projectName string
    projectPackage string
)

func init() {
    if Settings.Project != "" {
        projectPackage = Settings.Project
    }

    root, err := os.Getwd()
    if err != nil {
        log.Fatal(err)
    }
    projectName = filepath.Base(root)
}

func ProjectName() string {
    return projectName
}

func ProjectPackage() string {
    return projectPackage
}
```

Figura 46: Fichero de configuración pkg/config/project.go.

## 4.9 Contenedores base de UALTools

En esta sección se analizará cada uno de los Dockerfile que conforman las herramientas de la aplicación y se explicarán los aspectos más interesantes de cada uno de ellos.

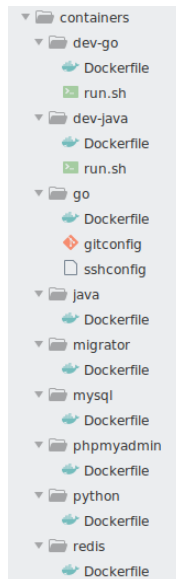


Figura 47: Árbol de contenedores de la aplicación.

## 4.9 Contenedores base de UALTools

### 4.9.1 dev-go

```
FROM golang:1.12

RUN apt-get update && \
    apt-get install -y zip unzip

RUN mkdir /home/container && \
    chmod 0777 /home/container && \
    mkdir /data && \
    chmod 0777 /data

ENV GOPATH /gotools
RUN go get -u github.com/cortesi/modd/cmd/modd
ENV GOPATH /go

COPY run.sh /opt/run.sh

ENV PATH $PATH:/gotools/bin

CMD ["/opt/run.sh"]
```

Figura 48: Dockerfile del contenedor dev-go.

Este Dockerfile usa la imagen de Go 1.12, después ejecuta una serie de comandos donde en primer lugar, actualiza el contenedor. Después, crea el directorio `/home/container` y `/data`, asignándole todos los permisos. De tal forma que se puedan evitar problemas a la hora de que el contenedor necesite realizar ciertas operaciones de escritura.

Después, se indica el `GOPATH` y se instala un paquete necesario para facilitar la ejecución del Go como servicio (de la forma que interesa para UALTools). La librería en cuestión es <https://github.com/cortesi/modd>.

Por último, se copia un script que será ejecutado al final del presente Dockerfile y se establecerá la variable de entorno `PATH` a `/gotools/bin`.

## 4.9 Contenedores base de UALTools

```
#!/bin/bash

set -eu

if [[ -z $WORKDIR ]]; then
    APP=$SERVICE
else
    APP=$(basename $WORKDIR)
fi

cd /workspace/$WORKDIR

echo ""
**/*.go /workspace/pkg/**/*.go {
    prep: go install ./cmd/$APP
}

/go/bin/$APP /etc/$APP/**/*.go {
    daemon +sigterm: $APP
}
"" > /tmp/modd.conf

modd -f /tmp/modd.conf
```

Figura 49: Script run.sh que se ejecuta al poner en marcha el contenedor.

Este Script se encarga de ubicar al contenedor en el WORKDIR, configura e instala la aplicación para más adelante poder ejecutarla con el comando ‘modd -f /tmp/modd.conf’

### 4.9.2 dev-java

```
FROM maven:3.3-jdk-8

RUN mkdir /home/container && \
    chmod 0777 /home/container && \
    mkdir /home/container/.m2 && \
    chmod 0777 /home/container/.m2 && \
    mkdir /data && \
    chmod 0777 /data

RUN groupadd --gid 1000 -r localgrp -o && \
    useradd --system --uid=1000 --gid=1000 --home-dir /home/container local1000 -o && \
    useradd --system --uid=1001 --gid=1000 --home-dir /home/container local1001 -o

ENV MAVEN_CONFIG /home/container/.m2

CMD ["mvn", "spring-boot:run"]
```

Figura 50: Dockerfile del contenedor dev-java.

Este es el Dockerfile del contenedor para entornos de desarrollo en Java. Utiliza la imagen de maven. Ya que se decidió que el contenedor pudiese ejecutar APIs, se optó que éstas fuesen en Spring y las dependencias se gestionasen con maven.

El contenedor crea los directorios /home/container y /data al igual que dev-go y además, crea un directorio en /home/container/.m2.



## 4.9 Contenedores base de UALTools

Posteriormente, copia al usuario del sistema Host y lo añade dentro del contenedor. Por último, establece una variable de entorno MAVEN\_CONFIG al directorio /home/container/.m2.

Por último, el contenedor ejecuta el comando ‘mvn spring-boot:run’

### 4.9.3 go

```
FROM golang:1.12
RUN apt-get update && \
    apt-get install -y zip unzip
RUN groupadd --gid 1000 -r localgrp -o && \
    useradd --system --uid=1000 --gid=1000 --home-dir /home/container local1000 -o && \
    useradd --system --uid=1001 --gid=1000 --home-dir /home/container local1001 -o
COPY gitconfig /home/container/.gitconfig
COPY gitconfig /root/.gitconfig
COPY sshconfig /home/container/.ssh/config
COPY sshconfig /root/.ssh/config
WORKDIR /workspace
```

Figura 51: Dockerfile del contenedor go.

Este es un contenedor un tanto especial, ya que necesita configurar sus propiedades ssh internas y cambiar el puntero de git para poder ejecutarlo sobre la máquina host. Además de añadir las propiedades del usuario de la máquina host en el contenedor.

### 4.9.4 java

```
FROM openjdk:8-jdk-alpine
WORKDIR /workspace
```

Figura 52: Dockerfile del contenedor java.

En este Dockerfile, simplemente se usa la imagen de openjdk y se establece el WORKDIR a /workspace.

### 4.9.5 python

```
FROM python:3
```

Figura 53: Dockerfile del contenedor python.

En el contenedor de Python simplemente se usa la imagen de Python

## 4.9 Contenedores base de UALTools

### 4.9.6 mysql

```
FROM mysql:5.7
ENV MYSQL_ROOT_PASSWORD dev-root
ENV MYSQL_USER dev-user
ENV MYSQL_PASSWORD dev-password
ENV MYSQL_DATABASE default
CMD ["--character-set-server=utf8mb4", "--collation-server=utf8mb4_bin"]
```

Figura 54: Dockerfile del contenedor mysql.

Este contenedor usa la imagen oficial de MySQL y configura una base de datos con unas credenciales por defecto y una configuración de codificación.

### 4.9.7 redis

```
FROM redis:5
```

Figura 55: Dockerfile del contenedor Redis.

Al igual que el de Python, este contenedor simplemente utiliza la imagen de Redis.

### 4.9.8 phpmyadmin

```
FROM phpmyadmin/phpmyadmin:latest
ENV PMA_HOST database
```

Figura 56: Dockerfile del contenedor phpmyadmin.

El Dockerfile de phpmyadmin configura una variable de entorno que permite establecer la conexión con la base de datos del entorno.

### 4.9.9 migrator

```
FROM golang:1.12
RUN go get -u github.com/altipla-consulting/migrator/cmd/init-migrator
RUN go get -u github.com/altipla-consulting/migrator/cmd/migrator
WORKDIR /workspace
```

Figura 57: Dockerfile del contenedor migrator.

Este Contenedor utiliza la imagen de Go e instala 2 herramientas del repositorio <https://github.com/altipla-consulting/migrator>. Estas herramientas permiten

## 4.10 Comandos

inicializar una base de datos para realizar migraciones (añadiendo una tabla para ello) y además ejecutan los ficheros de migración sobre la base de datos con un sencillo comando.

### 4.10 Comandos

Esta es una, si no la sección más importante dentro del presente TFG. En ella, se describirá de qué forma se han definido todos y cada uno de los comandos que el usuario puede ejecutar, y de qué forma interactúan los controladores con los contenedores y todos los aspectos anteriormente explicados para llevar a cabo la ejecución de dichos comandos.

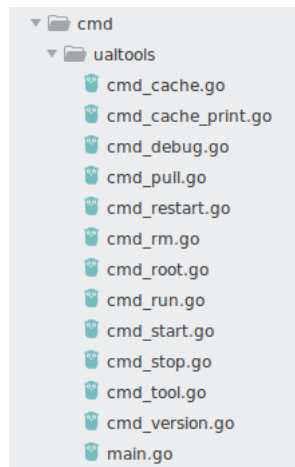


Figura 58: Árbol de ficheros del directorio **cmd/ualtools**.

#### 4.10.1 Librería <https://github.com/spf13/cobra>

Se usará la librería Cobra para establecer en el **main.go** un comando de entrada de la aplicación (que se encuentra CmdRoot) y ejecutar sobre él el método `.Execute()`. En el resto de ficheros del directorio **cmd/ualtools** se definirán el resto de comandos que, una vez ejecutada la aplicación, se detectarán automáticamente.

## 4.10 Comandos

### 4.10.2 main.go

```
package main

import (
    "os"
)

func main() {
    if err := CmdRoot.Execute(); err != nil {
        os.Exit(1)
    }
}
```

Figura 59: Punto de entrada de la aplicación.

Este fichero es el punto de entrada de la aplicación, directamente llama al método **CmdRoot.Execute()**, comentado en el punto anterior, que inicializa el comando de entrada de la aplicación, ‘ualtools ...’ que se mostrará en detalle a continuación.

### 4.10.3 cmd\_root.go

```
package main

import (
    "github.com/dvormagic/ualtools/pkg/update"
    "github.com/juju/errors"
    log "github.com/sirupsen/logrus"
    "github.com/spf13/cobra"
)

var debugApp bool

func init() {
    CmdRoot.PersistentFlags().BoolVarP(&debugApp, "debug", "d", false, "Activa el logging de depuración")
}

var CmdRoot = &cobra.Command{
    Use:     "ualtools",
    Short:   "Helper de desarrollo de herramientas de UALTools",
    SilenceUsage: true,
    PersistentPreRunE: func(cmd *cobra.Command, args []string) error {
        if debugApp {
            log.SetLevel(log.DebugLevel)
            log.Debug("DEBUG log level activated")
        }
        return nil
    },
}
```

Figura 60: Raíz de ejecución de comandos.

Aquí se define el comando de inicio de la aplicación ‘ualtools ...’. Como se puede apreciar, se le añade el flag ‘d’ o ‘debug’ (y se establece a false) para permitir al usuario activar el logging de depuración o no, haciendo uso de una variable booleana **debugApp**.

## 4.10 Comandos

Cabe destacar que el campo del comando `CmdRoot` **PersistentPreRunE**, en caso de que el logging de depuración se haya activado, se establecerá dicho nivel de depuración con la librería que se ha utilizado para los logs de UALTools, <https://github.com/sirupsen/logrus>

### 4.10.4 cmd\_start.go

Este es el comando más complejo y el que arranca los servicios de desarrollo. Este comando hará comprobaciones del fichero definido y configurado por el usuario `ualtools.yml`.

```
func init() {
    CmdRoot.AddCommand(CmdStart)
}

var CmdStart = &cobra.Command{
    Use: "start",
    Short: "Enciende un servicio de desarrollo",
    RunE: func(cmd *cobra.Command, args []string) error {
        return errors.Trace(startCommand(args))
    },
}
```

Figura 61: Inicio de fichero de comando start.

En primer lugar, se añade el comando a `CmdRoot` para que el usuario pueda continuar la secuencia `'ualtools start ...'`. Al ejecutarlo se hará una llamada al método privado **startCommand(args)**, dónde se le añadirán los argumentos que se le hayan pasado a continuación del comando anteriormente mencionado.

Antes de continuar explicando el método **startCommand**, se procederá a explicar el método **resolveDeps**

## 4.10 Comandos

```
func resolveDeps(args []string) ([]string, []string, error) {
    if len(args) == 0 {
        return nil, nil, nil
    }

    services := []string{}
    tools := []string{}
    for _, arg := range args {
        if config.Settings.IsService(arg) {
            services = append(services, arg)

            subservices, subtools, err := resolveDeps(config.Settings.Services[arg].Deps)
            if err != nil {
                return nil, nil, errors.Trace(err)
            }
            services = append(services, subservices...)
            tools = append(tools, subtools...)

            continue
        }

        if config.Settings.IsTool(arg) {
            tools = append(tools, arg)

            subservices, subtools, err := resolveDeps(config.Settings.Tools[arg].Deps)
            if err != nil {
                return nil, nil, errors.Trace(err)
            }
            services = append(services, subservices...)
            tools = append(tools, subtools...)

            continue
        }

        return nil, nil, errors.NotFoundf("service %s", arg)
    }

    services = collections.UniqueStrings(services)
    tools = collections.UniqueStrings(tools)

    return services, tools, nil
}
```

Figura 62: Método privado **resolveDeps**

Este método hace comprobaciones de si los argumentos que se han enviado son servicios o herramientas (haciendo uso del paquete `config` explicado en el punto 4.9.1, **Figura 43**) y las extrae de la configuración definida por el usuario `ualtools.yml`. Una vez hace la primera comprobación, vuelve a comprobar si en las deps del servicio o herramienta hay más herramientas o servicios. Cuando se comprueban los servicios y herramientas, se añaden a dos listas (una para cada tipo) que son las que devuelve finalmente el método.

## 4.10 Comandos

```
func startCommand(args []string) error {
    services, tools, err := resolveDeps(args)
    if err != nil {
        return errors.Trace(err)
    }

    for _, tool := range tools {
        containerDesc, err := containers.FindImage(config.Settings.Tools[tool].Container)
        if err != nil {
            return errors.Trace(err)
        }

        options := []docker.ContainerOption{
            docker.WithImage(docker.Image(containers.Repo, containerDesc.Image)),
            docker.WithDefaultNetwork(),
            docker.WithPersistence(),
            docker.WithNetworkAlias(tool),
        }
        options = append(options, containerDesc.Options...)

        for _, port := range config.Settings.Tools[tool].Ports {
            parts := strings.Split(port, ":")
            if len(parts) != 2 {
                return errors.Errorf("ports of tool %s", tool)
            }

            source, err := strconv.ParseInt(parts[0], 10, 64)
            if err != nil {
                return errors.NewNotValid(err, fmt.Sprintf("invalid port number: %s", parts[0]))
            }

            inside, err := strconv.ParseInt(parts[1], 10, 64)
            if err != nil {
                return errors.NewNotValid(err, fmt.Sprintf("invalid port number: %s", parts[1]))
            }

            options = append(options, docker.WithPort(source, inside))
        }

        for _, volume := range config.Settings.Tools[tool].Volumes {
            parts := strings.Split(volume, ":")
            if len(parts) != 2 {
                return errors.Errorf("volumes of tool %s", tool)
            }

            options = append(options, docker.WithVolume(parts[0], parts[1]))
        }

        container, err := docker.Container(tool, options...)
        if err != nil {
            return errors.Trace(err)
        }

        log.WithField("service", tool).Info("Start service")
        if err := container.Start(config.Settings.Tools[tool].Args...); err != nil {
            return errors.Trace(err)
        }
    }
}
```

Figura 63: Preparación de herramientas.

Una vez explicado el proceso de extraer los servicios y herramientas, se comienza por recorrer cada una de las herramientas. El primer paso al recorrer las herramientas, es el de buscar que la imagen se encuentre entre las imágenes de los contenedores definidos en el punto 4.6.

Si todo va bien, se definirá una colección de opciones funcionales (la técnica de Go que permitía añadir opcionalmente funciones que se ejecutaban internamente, ubicadas en el fichero `pkg/docker/container_options.go` y explicadas en el punto 4.5.4) dónde se añaden: **WithImage**, **WithDefaultNetwork**, **WithPersistence** y **WithNetworkAlias**. Después de añadir estas cuatro opciones funcionales, se añaden todas las que vienen por defecto desde el contenedor extraído de `container_options.go`.

Una vez añadidas las opciones, se comprueban los puertos y volúmenes definidos en el archivo de configuración, comprobando además que el puerto está

#### 4.10 Comandos

escrito en un formato correcto para finalmente, añadir las opciones funcionales **WithPort** o **WithVolume** en función de lo que corresponda. Cuando se han añadido todas las opciones funcionales, se usa el constructor del controlador de Docker para generar un contenedor con todas las opciones añadidas anteriormente y el nombre de la herramienta que corresponda. Por último, se lanza el comando **Start** para arrancar la herramienta (que crea el contenedor en caso de no existir). Una vez todas las herramientas se han iniciado/creado. Se procede a crear los servicios. Un servicio será básicamente una aplicación en Go o en Java que podrá interaccionar con distintas herramientas.



## 4.10 Comandos

```
watcher := docker.NewWatcher()
for _, service := range services {
    containerDesc, err := containers.FindImage(fmt.Sprintf("dev-%s", config.Settings.Services[service].Type))
    if err != nil {
        return errors.Trace(err)
    }

    options := []docker.ContainerOption{
        docker.WithImage(docker.Image(containers.Repo, containerDesc.Image)),
        docker.WithDefaultNetwork(),
        docker.WithPersistence(),
        docker.WithWorkdir(fmt.Sprintf("/workspace/%s", config.Settings.Services[service].Workdir)),
        docker.WithNetworkAlias(service),
        docker.WithEnv("PROJECT", config.Settings.Project),
        docker.WithEnv("WORKDIR", config.Settings.Services[service].Workdir),
        docker.WithEnv("SERVICE", service),
    }
    options = append(options, containerDesc.Options...)

    for _, port := range config.Settings.Services[service].Ports {
        parts := strings.Split(port, ":")
        if len(parts) != 2 {
            return errors.NotValidf("ports of service %s", service)
        }

        source, err := strconv.ParseInt(parts[0], 10, 64)
        if err != nil {
            return errors.NewNotValid(err, fmt.Sprintf("invalid port number: %s", parts[0]))
        }

        inside, err := strconv.ParseInt(parts[1], 10, 64)
        if err != nil {
            return errors.NewNotValid(err, fmt.Sprintf("invalid port number: %s", parts[1]))
        }

        options = append(options, docker.WithPort(source, inside))
    }

    for _, volume := range config.Settings.Services[service].Volumes {
        parts := strings.Split(volume, ":")
        if len(parts) != 2 {
            return errors.NotValidf("volumes of service %s", service)
        }

        options = append(options, docker.WithVolume(parts[0], parts[1]))
    }

    for k, v := range config.Settings.Services[service].Env {
        options = append(options, docker.WithEnv(k, v))
    }

    container, err := docker.Container(service, options...)
    if err != nil {
        return errors.Trace(err)
    }

    watcher.Run(service, container)
}

watcher.Wait()

return nil
}
```

Figura 64: Creación de watcher y preparación de servicios.

Primero se inicializa un watcher (aquel elemento que permitía que las goroutines se ejecutasen de una en una). Posteriormente, se comienzan a recorrer los servicios procediendo exactamente de la misma forma que se ha hecho con las herramientas, sólo que con distintas opciones funcionales y que al final, se añaden variables de entorno en caso de haberlas y en lugar de ejecutar el método Start del contenedor, se envía al método Run del watcher, que ejecutaba el comando ‘docker start ...’ por debajo en goroutines independientes.

## 4.10.5 cmd\_stop.go

Este es el fichero dedicado al comando stop.

```
package main

import (
    "github.com/juju/errors"
    log "github.com/sirupsen/logrus"
    "github.com/spf13/cobra"

    "github.com/dvormagic/ualtools/pkg/config"
    "github.com/dvormagic/ualtools/pkg/docker"
)

func init() {
    CmdRoot.AddCommand(CmdStop)
}

var CmdStop = &cobra.Command{
    Use: "stop",
    Short: "Apaga un servicio de desarrollo. Sin argumentos apaga todos los servicios.",
    RunE: func(cmd *cobra.Command, args []string) error {
        return errors.Trace(stopCommand(args))
    },
}

func stopCommand(args []string) error {
    if len(args) == 0 {
        for tool := range config.Settings.Tools {
            args = append(args, tool)
        }
    }

    for _, arg := range args {
        if !config.Settings.IsTool(arg) {
            return errors.NotFoundf("service %s", arg)
        }
    }

    for _, service := range args {
        container, err := docker.Container(service)
        if err != nil {
            return errors.Trace(err)
        }

        if running, err := container.Running(); err != nil {
            return errors.Trace(err)
        } else if !running {
            continue
        }

        log.WithField("service", service).Info("Stop service")
        if err := container.Stop(); err != nil {
            return errors.Trace(err)
        }
    }

    return nil
}
```

Figura 65: Fichero de comando para parar servicios.

Como se puede apreciar este fichero es bastante más sencillo que el del comando start. Básicamente, si no se le pasa ningún argumento (servicio o herramienta). Recorre todas las herramientas de configuración y las almacena como argumentos. Recorre todos los servicios y extrae su contenedor correspondiente con el método **docker.Container()**. Si el contenedor no está en ejecución continua con el siguiente servicio o herramienta, si está ejecutandose ejecuta el comando **container.Stop()**.

## 4.10 Comandos

### 4.10.6 cmd\_restart.go

Fichero del comando restart.

```
package main

import (
    "github.com/juju/errors"
    "github.com/spf13/cobra"
)

func init() {
    CmdRoot.AddCommand(CmdRestart)
}

var CmdRestart = &cobra.Command{
    Use: "restart",
    Short: "Reinicia un servicio de desarrollo.",
    RunE: func(cmd *cobra.Command, args []string) error {
        if len(args) == 0 {
            return errors.NotValidf("arguments required")
        }

        if err := stopCommand(args); err != nil {
            return errors.Trace(err)
        }
        if err := startCommand(args); err != nil {
            return errors.Trace(err)
        }

        return nil
    },
}
```

Figura 66: Fichero de comando para reiniciar servicios.

Este comando reutiliza los métodos de **stopCommand** y **startCommand** explicados en los puntos anteriores para reiniciar el servicio de desarrollo.

### 4.10.7 cmd\_rm.go

Fichero del comando rm.

## 4.10 Comandos

```
package main

import (
    "github.com/juju/errors"
    log "github.com/sirupsen/logrus"
    "github.com/spf13/cobra"

    "github.com/dvormagic/ualtools/pkg/config"
    "github.com/dvormagic/ualtools/pkg/docker"
)

func init() {
    CmdRoot.AddCommand(CmdRm)
}

var CmdRm = &cobra.Command{
    Use: "rm",
    Short: "Elimina un servicio de desarrollo. Sin argumentos elimina todos los servicios.",
    RunE: func(cmd *cobra.Command, args []string) error {
        if len(args) == 0 {
            for tool := range config.Settings.Tools {
                args = append(args, tool)
            }
            for service := range config.Settings.Services {
                args = append(args, service)
            }
        }

        for _, arg := range args {
            if !config.Settings.IsService(arg) && !config.Settings.IsTool(arg) {
                return errors.NotFoundf("service %s", arg)
            }
        }

        for _, service := range args {
            container, err := docker.Container(service)
            if err != nil {
                return errors.Trace(err)
            }

            if exists, err := container.Exists(); err != nil {
                return errors.Trace(err)
            } else if !exists {
                continue
            }

            if running, err := container.Running(); err != nil {
                return errors.Trace(err)
            } else if running {
                log.WithField("service", service).Info("Stop service")
                if err := container.Stop(); err != nil {
                    return errors.Trace(err)
                }
            }

            log.WithField("service", service).Info("Remove service")
            if err := container.Remove(); err != nil {
                return errors.Trace(err)
            }
        }

        return nil
    },
}
```

Figura 67: Fichero de comando para eliminar contenedores de servicios.

Si no hay argumentos, recopila todos los servicios y herramientas. Más adelante, recorre dichos servicios y herramientas, extrayendo su contenedor y en caso de que se esté ejecutando, llama al método **Stop()** del contenedor previamente para finalmente (y caso en el que el contenedor no se está ejecutando) llamar al método **Remove()**.

## 4.10.8 cmd\_tool.go

Este comando es el que se utiliza para usar herramientas de forma externa sin necesidad de crear entornos, interaccionando con el directorio dónde esté ubicado el sistema.

Previamente a la explicación del comando, se procede a explicar la función **createToolEntrypoint** que tendrá gran peso en el uso del comando.

```
func createToolEntrypoint(containerDesc containers.Container, tool, workdir string) func(cmd *cobra.Command, args []string) error {
    return func(cmd *cobra.Command, args []string) error {
        options := []docker.ContainerOption{
            docker.WithImage(docker.Image(containers.Repo, containerDesc.Image)),
            docker.WithDefaultNetwork(),
            docker.WithEnv("PROJECT", config.Settings.Project),
        }
        options = append(options, containerDesc.Options...)
        if workdir != "" {
            options = append(options, docker.WithWorkdir(fmt.Sprintf("/workspace/%s", workdir)))
        }

        container, err := docker.Container(fmt.Sprintf("tool-%s-%s", containerDesc.Image, tool), options...)
        if err != nil {
            return errors.Trace(err)
        }

        args = append([]string{tool}, args...)

        if err := container.Run(args...); err != nil {
            return errors.Trace(err)
        }

        return nil
    }
}
```

Figura 68: Método para crear el endpoint de una herramienta.

Esta función recibe un contenedor por argumento, una herramienta y el directorio de trabajo y retorna una función para ser ejecutada como comando por Cobra. En primer lugar, se inicializa una lista de opciones funcionales. Después, se añaden las opciones funcionales del contenedor a dicha lista. En caso de que el directorio de trabajo venga vacío, se establece uno por defecto. Después, se llama al constructor del controlador de contenedores para crear un contenedor con el tag **tool-(imagen)-(herramienta)** y las opciones funcionales. Se cogen los argumentos que se hayan añadido después del comando y se ejecuta el contenedor con dichos argumentos.

Ahora si, se procede a la explicación del comando.

## 4.10 Comandos

```
func init() {
    var CmdApp = &cobra.Command{
        Use: "app",
        Short: "Ejecuta una herramienta dentro de la carpeta de una aplicación",
    }
    CmdRoot.AddCommand(CmdApp)

    CmdsApp := map[string]*cobra.Command{}
    for name := range config.Settings.Services {
        var CmdAppService = &cobra.Command{
            Use: name,
            Short: fmt.Sprintf("Servicio %s", name),
        }
        CmdApp.AddCommand(CmdAppService)
        CmdsApp[name] = CmdAppService
    }

    for _, container := range containers.List() {
        for _, tool := range container.Tools {
            var CmdToolDirect = &cobra.Command{
                Use: tool,
                Short: fmt.Sprintf("Herramienta %s [%s]", tool, container.Image),
                DisableFlagParsing: true,
                DisableFlagsInUseLine: true,
                RunE: createToolEntrypoint(container, tool, ""),
            }
            CmdRoot.AddCommand(CmdToolDirect)

            var CmdToolDebug = &cobra.Command{
                Use: tool,
                Short: fmt.Sprintf("Herramienta %s [%s]", tool, container.Image),
                DisableFlagParsing: true,
                DisableFlagsInUseLine: true,
                RunE: createToolEntrypoint(container, tool, ""),
            }
            CmdDebug.AddCommand(CmdToolDebug)

            for name, service := range config.Settings.Services {
                var CmdAppServiceTool = &cobra.Command{
                    Use: tool,
                    Short: fmt.Sprintf("Herramienta %s [%s]", tool, container.Image),
                    DisableFlagParsing: true,
                    DisableFlagsInUseLine: true,
                    RunE: createToolEntrypoint(container, tool, service.Workdir),
                }
                CmdsApp[name].AddCommand(CmdAppServiceTool)
            }
        }
    }
}
```

Figura 69: Fichero de comando para ejecutar herramientas en directorios.

Este comando inicializa un mapa de string (herramienta o servicio) a comandos de Cobra. Posteriormente, recorre los servicios en caso de que se esté ejecutando el comando desde un directorio de un servicio, para añadirlos al mapa.

Después, recorre todos los contenedores y herramientas haciendo una llamada al método **List()** visto en el fichero de contenedores **pkg/containers/containers.go**, para crear un comando por cada herramienta, herramienta de depuración o servicio en caso de haberlo. Internamente, cada uno de estos comandos ejecuta el método descrito anteriormente **createToolEntrypoint**, con los parámetros adaptados a cada una de las herramientas.

De esta forma, cada vez que se haga una llamada del tipo 'ualtools python

## 4.10 Comandos

hello\_world.py', UALTools lo reconocerá ya que se ha establecido el entrypoint a raíz del fichero de contenedores.

### 4.10.9 cmd\_cache\_print.go

Este comando es meramente informativo.

```
package main

import (
    "fmt"

    "github.com/spf13/cobra"

    "github.com/dvormagic/ualtools/pkg/config"
)

func init() {
    CmdCache.AddCommand(CmdCachePrint)
}

var CmdCachePrint = &cobra.Command{
    Use: "print",
    Short: "Imprime el directorio donde los artefactos de las herramientas están cacheados",
    RunE: func(cmd *cobra.Command, args []string) error {
        fmt.Printf("%s/.ualtools/cache-%s\n", config.Home(), config.ProjectName())
        return nil
    },
}
```

Figura 70: Fichero de comando para mostrar directorio de caché de herramientas.

Al llamarlo, imprimirá la localización de dónde están cacheados los artefactos de las herramientas.

### 4.10.10 cmd\_debug.go

Este es el fichero del comando para activar el logging de depuración

```
package main

import (
    log "github.com/sirupsen/logrus"
    "github.com/spf13/cobra"
)

var CmdDebug = &cobra.Command{
    Use: "debug",
    Short: "Activa el modo depuración de las herramientas",
    PersistentPreRun: func(cmd *cobra.Command, args []string) {
        log.SetLevel(log.DebugLevel)
        log.Debug("DEBUG log level activated")
    },
}

func init() {
    CmdRoot.AddCommand(CmdDebug)
}
```

Figura 71: Fichero de comando para activar logging de depuración.

## 4.10 Comandos

Al ejecutarlo, se establecerá el nivel de depuración con la librería que se ha utilizado para los logs de UALTools, <https://github.com/sirupsen/logrus>

### 4.10.11 cmd\_run.go

Este fichero representa al comando run, que accede al contenedor de una herramienta y permite ejecutar comandos manualmente dentro del mismo.

```
package main

import (
    "fmt"

    "github.com/juju/errors"
    "github.com/spf13/cobra"

    "github.com/dvormagic/ualtools/pkg/containers"
    "github.com/dvormagic/ualtools/pkg/docker"
)

func init() {
    CmdRoot.AddCommand(CmdRun)
    for _, container := range containers.List() {
        var CmdContainer = &cobra.Command{
            Use:           container.Image,
            Short:         fmt.Sprintf("Contenedor %s", container.Image),
            DisableFlagParsing: true,
            DisableFlagsInUseLine: true,
            RunE:         createRunEntrypoint(container),
        }
        CmdRun.AddCommand(CmdContainer)
    }
}

var CmdRun = &cobra.Command{
    Use: "run",
    Short: "Ejecuta un comando manualmente dentro de un contenedor de herramienta concreto.",
}

func createRunEntrypoint(containerDesc containers.Container) func(cmd *cobra.Command, args []string) error {
    return func(cmd *cobra.Command, args []string) error {
        options := []docker.ContainerOption{
            docker.WithImage(docker.Image(containerDesc.Repo, containerDesc.Image)),
            docker.WithDefaultNetwork(),
        }
        options = append(options, containerDesc.Options...)

        container, err := docker.Container(fmt.Sprintf("run-%s", containerDesc.Image), options...)
        if err != nil {
            return errors.Trace(err)
        }

        if err := container.Run(args...); err != nil {
            return errors.Trace(err)
        }

        return nil
    }
}
```

Figura 72: Fichero de comando para acceder a contenedores para ejecutar comandos.

El método **createRunEntrypoint** se encarga de devolver una función que podrá ejecutar la librería Cobra (al igual que en 'ualtools tools ...'), define



## 4.10 Comandos

las opciones funcionales, añade las opciones funcionales del contenedor que corresponda y genera un contenedor con el constructor del controlador de contenedores. Por último, ejecuta el método **Run()** del controlador.

Volviendo a la definición de los comandos (principio del fichero), se recorre la lista de contenedores para crear cada uno de los comandos por cada uno de los contenedores y para finalmente, añadirse los al comando run. De tal forma que finalmente, se podría ejecutar por ejemplo ‘ualtools run python’, se accedería a la consola de python y se podrían ejecutar sentencias de python (entre otros ejemplos).

### 4.10.12 cmd\_pull.go

Este fichero es el que controla el comando de descarga y actualización de imágenes.

```
package main

import (
    "fmt"

    "github.com/juju/errors"
    log "github.com/sirupsen/logrus"
    "github.com/spf13/cobra"

    "github.com/dvormagic/ualtools/pkg/containers"
    "github.com/dvormagic/ualtools/pkg/docker"
)

func init() {
    CmdRoot.AddCommand(CmdPull)
}

var CmdPull = &cobra.Command{
    Use: "pull",
    Short: "Descarga y actualiza forzosamente las imágenes de los contenedores de herramientas.",
    RunE: func(cmd *cobra.Command, args []string) error {
        for _, image := range containers.Images() {
            log.WithField("image", image).Info("Download image")

            image := docker.Image("eu.gcr.io", fmt.Sprintf("ual-tools/%s", image))
            if err := image.Pull(); err != nil {
                return errors.Trace(err)
            }
        }
        return nil
    },
}
```

Figura 73: Fichero de comando para actualizar imágenes de contenedores.

Una vez se ejecuta el comando ‘ualtools pull’, se recorren todas las imágenes y gracias al controlador de imágenes, se inicializa cada una de ellas para posteriormente ejecutar el método **Pull** y descargarlas una a una en local.

## 4.11 Docker compose y pruebas del entorno

Ya se ha explicado cómo está estructurado el proyecto, sus paquetes más relevantes y cómo interaccionan entre ellos. Sólo falta explicar de qué forma se han ido probando los distintos comandos e interacciones con los distintos contenedores.

Se ha preparado el entorno para descargar las imágenes desde el container registry del proyecto en Google Cloud Platform. Evidentemente, no es la solución más óptima de cara a hacer pruebas de los distintos contenedores.

Para ello se ha hecho uso de docker-compose, el fichero es el siguiente:

```
version: '2'
services:
  dev-go:
    image: eu.gcr.io/ual-tools/dev-go:latest
    build:
      context: containers/dev-go

  dev-java:
    image: eu.gcr.io/ual-tools/dev-java:latest
    build:
      context: containers/dev-java

  go:
    image: eu.gcr.io/ual-tools/go:latest
    build:
      context: containers/go

  java:
    image: eu.gcr.io/ual-tools/java:latest
    build:
      context: containers/java

  python:
    image: eu.gcr.io/ual-tools/python:latest
    build:
      context: containers/python

  migrator:
    image: eu.gcr.io/ual-tools/migrator:latest
    build:
      context: containers/migrator

  redis:
    image: eu.gcr.io/ual-tools/redis:latest
    build:
      context: containers/redis

  mysql:
    image: eu.gcr.io/ual-tools/mysql:latest
    build:
      context: containers/mysql

  phpmyadmin:
    image: eu.gcr.io/ual-tools/phpmyadmin:latest
    build:
      context: containers/phpmyadmin
```

Figura 74: Fichero docker-compose del proyecto UALTools.

Con docker-compose, se define el conjunto de servicios o contenedores que se desea poner en marcha. Una vez definidos, para cada servicio, se asignan 2

#### 4.11 Docker compose y pruebas del entorno

valores.

- **image:** El tag que tiene el contenedor, para todos los casos **eu.grc.io/ual-tools/(contenedor):latest**. Ya que los que se descarguen de producción tendrán el mismo tag e interesa que se sobrescriba en caso de querer probar ciertos cambios de la aplicación.
- **build:** Dentro contiene el valor **context**, y básicamente es el sitio dónde se ubica el Dockerfile correspondiente a la imagen que se va a construir.

Para la construcción de un contenedor, basta con ejecutar por ejemplo el comando ‘docker-compose build python’, este comando, estará ejecutando por debajo ‘docker build -t python -f Dockerfile containers/python && docker tag foo eu.grc.io/ual-tools/python:latest && docker tag python eu.grc.io/ual-tools/python:123456’, que básicamente hace lo explicado anteriormente en la estructura del docker-compose.

De igual forma, si se ejecuta el comando ‘docker-compose build’, se lanzarán todos los comandos simultáneamente.

```
david@altiplaconsulting:~/proyectos/ualtools$ docker-compose build
Building dev-go
Step 1/9 : FROM golang:1.12
--> 8a91d9dc9d09
Step 2/9 : RUN apt-get update && apt-get install -y zip unzip
--> Using cache
--> 52e9c178e61b
Step 3/9 : RUN mkdir /home/container && chmod 0777 /home/container && mkdir /data && chmod 0777 /data
--> Using cache
--> dfdee9334b58
Step 4/9 : ENV GOPATH /gotools
--> Using cache
--> afd739c3bid1
Step 5/9 : RUN go get -u github.com/cortest/modd/cmd/modd
--> Using cache
--> c8136c6e077e
Step 6/9 : ENV GOPATH /go
--> Using cache
--> f7cc10e7e3d8
Step 7/9 : COPY run.sh /opt/run.sh
--> Using cache
--> 3dbaca6727bb
Step 8/9 : ENV PATH $PATH:/gotools/bin
--> Using cache
--> ffd7648a5ff3
Step 9/9 : CMD ["/opt/run.sh"]
--> Using cache
--> 481c4dff43e6
Successfully built 481c4dff43e6
Successfully tagged eu.grc.io/ual-tools/dev-go:latest
Building dev-java
Step 1/5 : FROM maven:3.3-jdk-8
--> 9997d8483b2f
Step 2/5 : RUN mkdir /home/container && chmod 0777 /home/container && mkdir /home/container/.m2 && chmod 0777 /home/container/.m2
--> Using cache
--> 1b9b7565913a
Step 3/5 : RUN groupadd --gid 1000 -r localgrp -o && useradd --system --uid=1000 --gid=1000 --home-dir /home/container local1000 -o &&
```

Figura 75: Ejecución comando ‘docker-compose build’.

Bien, una vez están montados todos los contenedores en local, es momento de compilar la aplicación. En este caso, ya que ha sido desarrollada con el sistema operativo Ubuntu Mate, simplemente será necesario ir al terminal, ubicarse en el directorio del proyecto ualtools y ejecutar el comando ‘go build -o /bin/ualtools ./cmd/ualtools’.

## 4.12 Despliegue

```
david@altiplaconsulting ~/projects/ualtools master go build -o ~/bin/ualtools ./cmd/ualtools
david@altiplaconsulting ~/projects/ualtools master ls ~/bin
ci stern ualtools
```

Figura 76: Compilación del programa UALTools con Go.

En el directorio de ficheros binarios de la carpeta personal del autor ya aparece el fichero binario ‘ualtools’, para comprobar que efectivamente se ha instalado, basta con escribir en la terminal el comando ‘ualtools’:

```
david@altiplaconsulting ~/projects/ualtools master ualtools
Helper de desarrollo de herramientas de UALTools

Usage:
  ualtools [command]

Available Commands:
  cache      Manage the tools local cache.
  debug      Activa el modo depuración de las herramientas
  go          Herramienta go [go]
  gofmt      Herramienta gofmt [go]
  help       Help about any command
  init-migrator Herramienta init-migrator [migrator]
  java       Herramienta java [java]
  javac      Herramienta javac [java]
  migrator   Herramienta migrator [migrator]
  mysql      Herramienta mysql [mysql]
  pull       Descarga y actualiza forzosamente las imágenes de los contenedores de herramientas.
  python     Herramienta python [python]
  redis-cli  Herramienta redis-cli [redis]
  restart    Reinicia un servicio de desarrollo.
  rm         Elimina un servicio de desarrollo. Sin argumentos elimina todos los servicios.
  run        Ejecuta un comando manualmente dentro de un contenedor de herramienta concreto.
  start      Enciende un servicio de desarrollo
  stop       Apaga un servicio de desarrollo. Sin argumentos apaga todos los servicios.
  version    Imprime la versión de la herramienta.

Flags:
  -d, --debug  Activa el logging de depuración
  -h, --help   help for ualtools

Additional help topics:
  ualtools app  Ejecuta una herramienta dentro de la carpeta de una aplicación
```

Figura 77: Ejecución del programa UALTools.

Finalmente, se hicieron las pruebas pertinentes de cada uno de los servicios y funcionalidades, ajustando los contenedores que lo necesitasen y consiguiendo que el sistema funcione correctamente y como el autor deseaba. En el apartado de **5. Resultados** se mostrará el uso de la aplicación en todas sus variantes.

## 4.12 Despliegue

Para poner en producción la aplicación, simplemente se ha subido el fichero binario generado a storage para todos los sistemas operativos. Para las distintas compilaciones se ejecutará:

- **Linux:** ‘go build -o /linux/ualtools ./cmd/ualtools’
- **Mac:** ‘env GOOS=darwin GOARCH=amd64 go build -o /mac/ualtools ./cmd/ualtools’

## 4.12 Despliegue

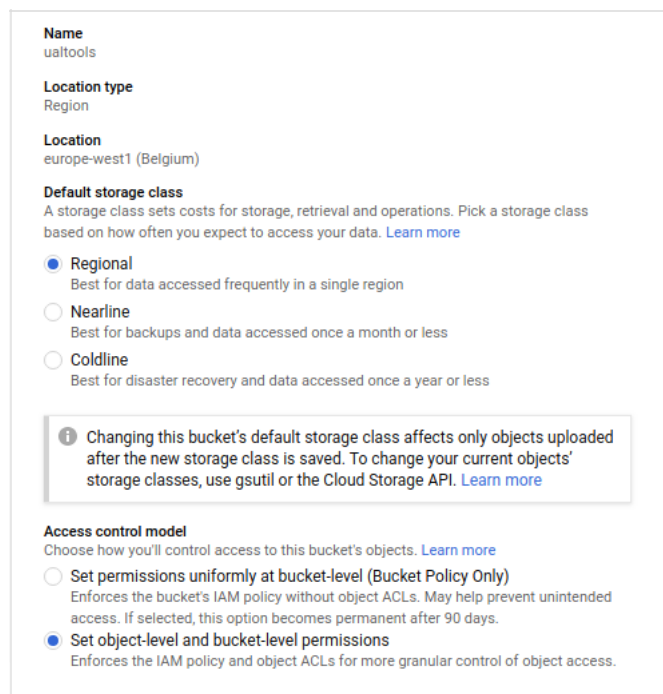
- **Windows:** `'env GOOS=windows GOARCH=amd64 go build -o /windows/ualtools.exe ./cmd/ualtools'`

Una vez ejecutados dichos comandos se ejecutarían los siguientes comandos para la subida a storage de los binarios:

- `'gsutil -h 'Cache-Control: no-cache' cp /linux/ualtools gs://ualtools/linux/ualtools'`
- `'gsutil -h 'Cache-Control: no-cache' cp /mac/ualtools gs://ualtools/mac/ualtools'`
- `'gsutil -h 'Cache-Control: no-cache' cp /windows/ualtools.exe gs://ualtools/windows/ualtools.exe'`

Para poder subir dichos ficheros a Storage, es necesario previamente habilitar un Bucket y autorizar la cuenta de administrador de Google Cloud Platform desde la máquina de desarrollo.

El bucket tiene las siguientes características:



The screenshot shows the configuration page for a Google Cloud Storage bucket named 'ualtools'. The configuration is set to 'Region' in the 'europe-west1 (Belgium)' location. The 'Default storage class' is set to 'Regional', which is described as 'Best for data accessed frequently in a single region'. There are also options for 'Nearline' and 'Coldline'. A warning message states: 'Changing this bucket's default storage class affects only objects uploaded after the new storage class is saved. To change your current objects' storage classes, use gsutil or the Cloud Storage API.' The 'Access control model' is set to 'Set object-level and bucket-level permissions', which enforces IAM policy and object ACLs.

**Name**  
ualtools

**Location type**  
Region

**Location**  
europe-west1 (Belgium)

**Default storage class**  
A storage class sets costs for storage, retrieval and operations. Pick a storage class based on how often you expect to access your data. [Learn more](#)

☒ **Regional**  
Best for data accessed frequently in a single region

☐ **Nearline**  
Best for backups and data accessed once a month or less

☐ **Coldline**  
Best for disaster recovery and data accessed once a year or less

**Changing this bucket's default storage class affects only objects uploaded after the new storage class is saved. To change your current objects' storage classes, use gsutil or the Cloud Storage API. [Learn more](#)**

**Access control model**  
Choose how you'll control access to this bucket's objects. [Learn more](#)

☐ **Set permissions uniformly at bucket-level (Bucket Policy Only)**  
Enforces the bucket's IAM policy without object ACLs. May help prevent unintended access. If selected, this option becomes permanent after 90 days.

☒ **Set object-level and bucket-level permissions**  
Enforces the IAM policy and object ACLs for more granular control of object access.

Figura 78: Características Bucket de Storage para subida de binarios.

## 4.12 Despliegue

Buckets / ualtools / linux

<input type="checkbox"/>	Name	Size	Type	Storage class	Last modified	Public access	Encryption	Retention expiry date	Holds
<input type="checkbox"/>	ualtools	11.07 MB	application/octet-stream	Regional	28/08/2019, 23:21:44 UTC+2	Public	Google-managed key	-	None

Figura 79: Binario Linux subido a Storage

Estos ficheros son los necesarios para poder instalar/utilizar UALTools, sus rutas son:

- **Linux:** <https://storage.googleapis.com/ualtools/linux/ualtools>
- **Windows:** <https://storage.googleapis.com/ualtools/windows/ualtools.exe>
- **MAC:** <https://storage.googleapis.com/ualtools/mac/ualtools>

Después de subir los binarios a Storage, de igual forma será necesario habilitar el servicio de container registry, configurarlo y subir todas las imágenes a storage.

Para la subida de los contenedores a storage, se ha ejecutado desde el directorio del proyecto el siguiente script bash:

```
#!/bin/bash

set -eu

function run {
    echo
    echo " x $1"
    bash -c "$1"
}

function docker-build-autotag {
    run "docker build -t container -f $2 $3"

    HASH=$(docker image inspect container -f '{{.Id}}' | cut -d ':' -f 2)
    VERSION=${HASH:0:12}

    run "docker tag container $1:latest"
    run "docker tag container $1:$VERSION"

    run "docker push $1:latest"
    run "docker push $1:$VERSION"
}

for FILE in containers/*/Dockerfile; do
    APP=$(basename $(dirname $FILE))
    docker-build-autotag eu.gcr.io/ual-tools/$APP containers/$APP/Dockerfile containers/$APP
done
```

Figura 80: Script de subida de contenedores.

Posteriormente, se comprueba que las imágenes se han sincronizado correctamente.

## 4.12 Despliegue

ual-tools

All hostnames

Name ^	Hostname	Visibility
dev-go	eu.gcr.io	Public
dev-java	eu.gcr.io	Public
dev-python	eu.gcr.io	Public
go	eu.gcr.io	Public
java	eu.gcr.io	Public
migrator	eu.gcr.io	Public
mongo	eu.gcr.io	Public
mysql	eu.gcr.io	Public
phpmyadmin	eu.gcr.io	Public
python	eu.gcr.io	Public
redis	eu.gcr.io	Public

Figura 81: Imágenes en container registry.

## 5 Resultados

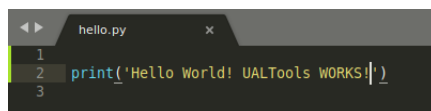
En este capítulo se abordará el funcionamiento de cada una de las funcionalidades de UALTools.

### 5.1 Ejemplo de uso de herramientas de UALTools

Para ilustrar este caso, se va a hacer uso de la herramienta Python, ya que permite interactuar con ella de distintas formas.

#### 5.1.1 Ejecución de scripts de Python

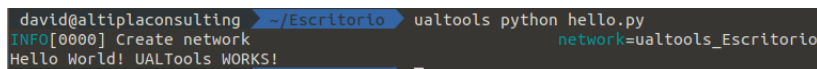
Será necesario un script .py, en este caso, se ha creado el script **hello.py**, que simplemente imprimirá por consola un "Hello World!".



```
1
2 print('Hello World! UALTools WORKS!')
3
```

Figura 82: hello.py.

Una vez el fichero está listo, desde el terminal se ejecuta el comando 'ualtools python hello.py':



```
david@altiplaconsulting ~/Escritorio ualtools python hello.py
INFO[0000] Create network
Hello World! UALTools WORKS!
network=ualtools_Escritorio
```

Figura 83: Hello World en python.

Al ser la ejecución de una herramienta, se creará una red con el nombre del directorio dónde se está ejecutando, y posteriormente la herramienta python ejecutará el código del script. Este caso es aplicable a ejecutar scripts en Go y en Java.

#### 5.1.2 Ejecución de test JUnit de Java

Aquí se va a mostrar como ejecutar un test sencillo de JUnit con Java. Para ello, se creará un directorio javaexample dónde añadiremos los ficheros de java (Abc y AbcExample) y además se añadirán las dependencias de JUnit (ya que son librerías externas). Los ficheros Abc y AbcTest son básicamente un sumador y la comprobación de la suma:



## 5.1 Ejemplo de uso de herramientas de UALTools

```
class Abc {  
    public int add(int a, int b) {  
        return a+b;  
    }  
}
```

Figura 84: Fichero Abc.java

```
import org.junit.After;  
import org.junit.AfterClass;  
import org.junit.Before;  
import org.junit.BeforeClass;  
import org.junit.Test;  
import static org.junit.Assert.*;  
  
public class AbcTest {  
    @Test  
    public void testAdd() {  
        System.out.println("add");  
        int a = 3;  
        int b = 5;  
        Abc instance = new Abc();  
        int expResult = 8;  
        int result = instance.add(a, b);  
        assertEquals(expResult, result);  
    }  
}
```

Figura 85: Fichero AbcTest.java

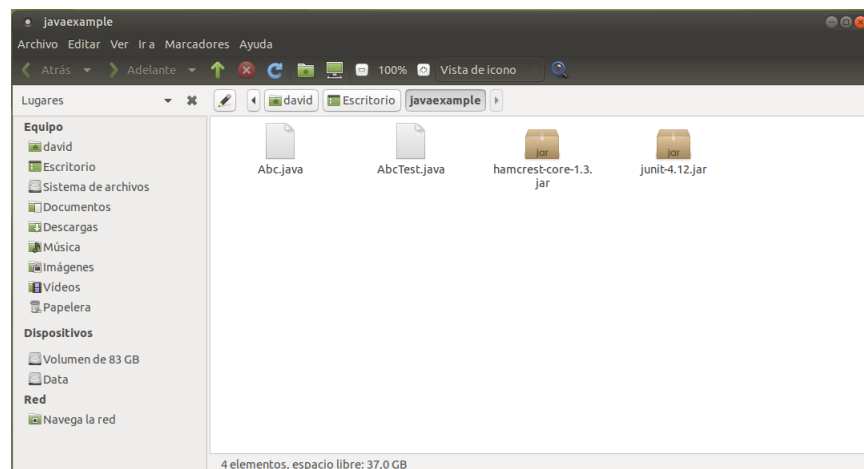


Figura 86: Directorio javaexample.

Ahora desde el terminal, será necesario compilar los archivos .java, para generar los .class y poder ejecutarlos posteriormente, para ello, se ejecutará el comando 'ualtools javac -cp junit-4.12.jar:. AbcTest.java'

## 5.1 Ejemplo de uso de herramientas de UALTools

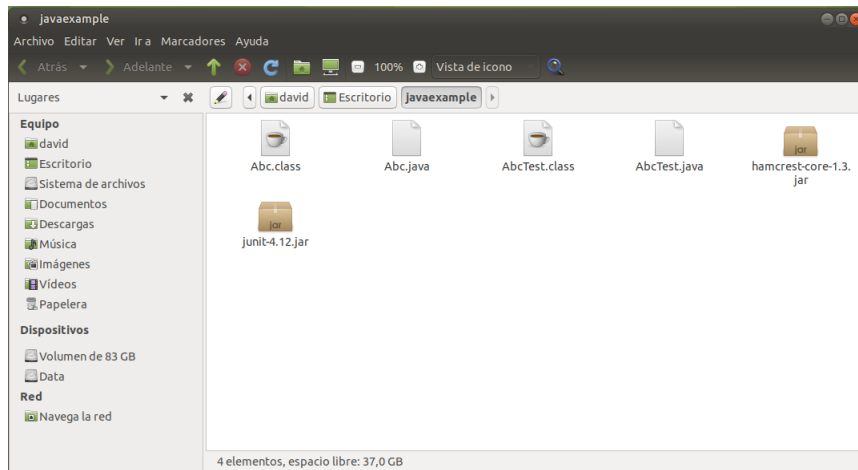


Figura 87: Directorio javaexample con ficheros compilados.

Como se puede apreciar, ahora han aparecido los ficheros compilados **Abc.class** y **AbcTest.class**. El siguiente paso es ejecutar el test unitario, ejecutando el comando 'ualtools java -cp junit-4.12.jar:hamcrest-core-1.3.jar:. org.junit.runner.JUnitCore AbcTest':

```
david@altiplaconsulting ~/Escritorio/javaexample ualtools java -cp junit-4.12.jar:hamcrest-core-1.3.jar:. org.junit.runner.JUnitCore AbcTest
JUnit version 4.12
.add

Time: 0.002
OK (1 test)
```

Figura 88: Ejecución test JUnit con UALTools.

Finalmente, se ejecutarán los test unitarios con el resultado esperado.

### 5.1.3 Ejecución de comandos desde dentro de un contenedor

Ahora se va a mostrar un ejemplo de cómo acceder a un contenedor de Python (se puede hacer con el resto de herramientas) y ejecutar comandos desde dentro.

```
david@altiplaconsulting ~/Escritorio/javaexample ualtools run python
Python 3.7.4 (default, Aug 14 2019, 12:09:51)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello World, UALTools works inside a Python container tool!!')
Hello World, UALTools works inside a Python container tool!!
>>>
```

Figura 89: Ejecución comandos dentro de contenedor Python.

Una vez se ejecuta el comando 'ualtools run python', se abrirá el terminal

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

desde dentro del propio contenedor, y permitirá ejecutar comandos. En este caso, se ha ejecutado otro "Hello World".

### 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

Esta sección, trata sobre cómo poner en marcha un servicio de desarrollo de UALTools, y muestra dos ejemplos ilustrativos que exponen el punto fuerte de la aplicación.

#### 5.2.1 Estructura del proyecto de ejemplo

El proyecto de ejemplo que se ha creado recibe el nombre de **example**, éste es el árbol de directorios:

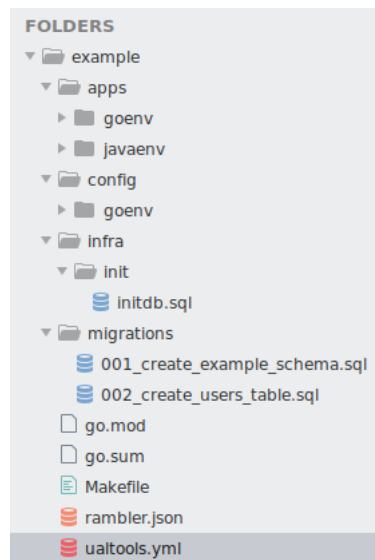


Figura 90: Directorio example.

En él se distinguen los directorios:

- **apps**, que es dónde están las dos aplicaciones o servicios en Go y Java, llamados goenv y javaenv respectivamente.
- **config**, que es el directorio dónde se añaden los ficheros de configuración de goenv.
- **infra**, que básicamente contiene un fichero initdb.sql creado para permitir la conexión de la base de datos MySQL entre distintos contenedores, en dicho fichero se ejecuta la sentencia SQL 'GRANT ALL ON \*.\* TO

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

'dev-user' (todo usuario que se logee con el nombre 'dev-user' tendrá todos los permisos sin importar la procedencia).

- **migrations**, que pasará a explicarse en el próximo punto y por último, y el núcleo del funcionamiento de los servicios de UALTools, el fichero **ualtools.yml**.

```
project: example

services:
  goenv:
    type: go
    workdir: apps/goenv
    deps:
      - redis
    ports:
      - "8000:8000"
    volumes:
      - "./config/goenv:/etc/goenv"

  javaenv:
    type: java
    workdir: apps/javaenv
    deps:
      - database
    ports:
      - "8080:8080"

tools:
  database:
    container: mysql
    ports:
      - "3306:3306"
    volumes:
      - "./infra/init:/docker-entrypoint-initdb.d"

  phpmyadmin:
    container: phpmyadmin
    ports:
      - "5000:80"

  redis:
    container: redis
    ports:
      - "6379:6379"
```

Figura 91: ualtools.yml

En este fichero, se definen los siguientes servicios y herramientas:

- **goenv**: Es de tipo go, se encuentra ubicado en apps/goenv, tiene como dependencia la herramienta redis, está expuesta en el puerto 8000 y expone un volumen de configuración config/goenv en etc/goenv dentro del contenedor.
- **javaenv**: Es de tipo Java, está ubicado en apps/javaenv, tiene la dependencia la herramienta database (nombre que se le ha asignado a la herramienta MySQL) y está expuesto en el puerto 8080.

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

- **database:** Esta herramienta usa el contenedor de mysql, está expuesto en el puerto 3306 y expone el fichero de infra/init comentado anteriormente dentro del contenedor, para que MySQL se configure internamente.
- **phpmyadmin:** Esta herramienta utiliza el contenedor de phpmyadmin y simplemente se expone en el puerto 5000.
- **redis:** Esta herramienta usa el contenedor de Redis y se expone en el puerto 6379.

### 5.2.2 Ejecución de migraciones

Para la ejecución de migraciones, es importante mostrar primero el fichero Makefile del proyecto example:

```
GOFILES = $(shell find . -type f -name "*.go" -not -path "./vendor/**")

.PHONY: migrations

gofmt:
    @gofmt -w $(GOFILES)
    @gofmt -r '&a{' -> new(a)' -w $(GOFILES)

initdb:
    ualtools init-migrator -user dev-user -password dev-password -address database

migrations:
    ualtools migrator -user dev-user -password dev-password -address database -directory migrations
```

Figura 92: Makefile

En él se muestran los comandos **gofmt** (para dar formato a los ficheros de Go), **initdb**, que ejecuta el comando ‘ualtools init-migrator -user dev-user -password dev-password -address database’, que consiste en decirle a la herramienta de migraciones de ualtools el usuario y la contraseña, para crear una base de datos “migrator” que será necesaria para la ejecución de migraciones. Aquí la comprobación en phpmyadmin (previamente, es necesario arrancarla con ‘ualtools start phpmyadmin’ desde el directorio example.

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

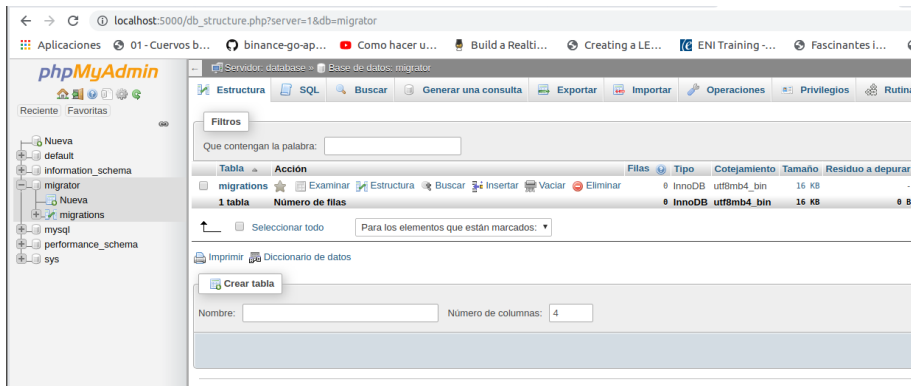


Figura 93: Schema de migrator.

Como se puede apreciar, en la ruta localhost:5000, entre las bases de datos, aparece una llamada migrator con la tabla migrations, que es la que se ha ejecutado al lanzar el comando 'make initdb'. El siguiente comando de Makefile es **migrations**, éste es el encargado de ejecutar las sentencias SQL ubicadas en el directorio **migrations** de UALTools. El directorio migrations, contiene dos ficheros:

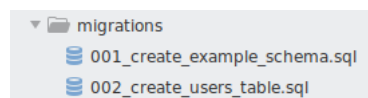


Figura 94: Ficheros de migración.

Aparecen 2 ficheros, uno para crear la base de datos de example y otro para crear una tabla users (su uso se verá explicado en el siguiente punto).

```
USE information_schema;  
CREATE SCHEMA example;
```

Figura 95: Creación de base de datos example.

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

```
USE example;

CREATE TABLE users (
  id INT(11) NOT NULL AUTO INCREMENT,
  name VARCHAR(191) NOT NULL,
  lastname VARCHAR(191) NOT NULL,
  email VARCHAR(191) NOT NULL,

  revision INT(11) NOT NULL DEFAULT 0,

  PRIMARY KEY(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

Figura 96: Creación de tabla users.

Una vez se ejecuta el comando **make migrations**:

```
david@altiplaconsulting:~/proyectos/ualtools$ make migrations
ualtools migrator -user dev-user -password dev-password -address database -directory migrations
INFO[0000] Connect to remote database           dir="/dev-user:/dev-password@tcp(database)/migrator?parseTime=true&charset=utf8mb4&collation=utf8mb4_bin"
INFO[0000] Apply migration                     database=information_schema name=001_create_example_schema.sql
INFO[0000] Connect to remote database           dir="/dev-user:/dev-password@tcp(database)/information_schema?parseTime=true&charset=utf8mb4&collation=utf8mb4_bin"
INFO[0000] Migration applied successfully!      database=example name=002_create_users_table.sql
INFO[0000] Connect to remote database           dir="/dev-user:/dev-password@tcp(database)/example?parseTime=true&charset=utf8mb4&collation=utf8mb4_bin"
INFO[0000] Migration applied successfully!
```

Figura 97: Ejecución comando make migrations.

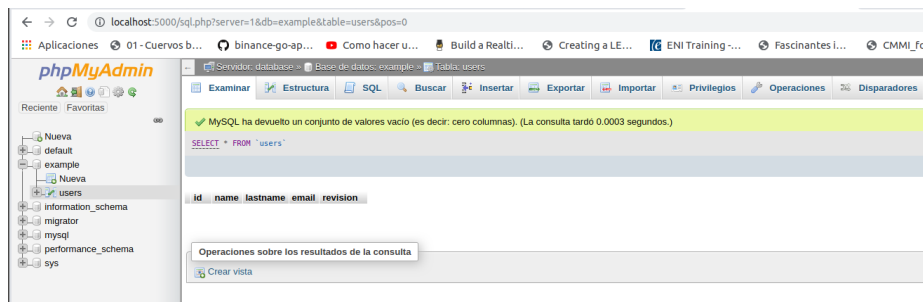


Figura 98: Tabla de usuarios creada vía migraciones.

Se puede comprobar con phpmyadmin que se ha creado la base de datos y la tabla esperadas.

### 5.2.3 Ejemplo API REST con Java y MySQL

Este punto expone la creación de una API REST con Java, que hace operaciones MySQL con la tabla creada en el punto anterior, de la misma forma, se podría hacer con Go.

La API que se ha creado está hecha con Spring, y permite gestionar los usuarios de la tabla que se ha creado anteriormente. En este punto, se expondrá el funcionamiento de la misma como servicio, sin profundizar en el código del mismo.

En primer lugar, será necesario ejecutar el comando ‘ualtools start javaenv’:

```

root@kali:~/http://kali:~# ./usr/bin/to/example ualtools start javaenv
[INFO][0000] Start service service=database
[INFO][0000] Start service service=javaenv
^[[A[INFO][0001] (javaenv) [INFO] Scanning for projects...
[INFO][0002] (javaenv) Downloading: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-parent/2.1.7.RELEASE/spring-boot-starter-parent-2.1.7.RELEASE.pom
[INFO][0002] (javaenv) 3/10 KB
[INFO][0002] (javaenv) 5/10 KB
[INFO][0002] (javaenv) 8/10 KB
[INFO][0002] (javaenv) 10/10 KB
[INFO][0002] (javaenv) Downloaded: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-parent/2.1.7.RELEASE/spring-boot-starter-parent-2.1.7.RELEASE.pom (10 KB at 33.7 KB/sec)
[INFO][0002] (javaenv) Downloading: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-dependencies/2.1.7.RELEASE/spring-boot-dependencies-2.1.7.RELEASE.pom
[INFO][0002] (javaenv) 3/119 KB
[INFO][0002] (javaenv) 5/119 KB
[INFO][0002] (javaenv) 8/119 KB
[INFO][0002] (javaenv) 11/119 KB
[INFO][0002] (javaenv) 13/119 KB

```

```
2220/2255 KB   aenv) 2212/2255 KB  
INFO[0021] (javaenv) 2224/2255 KB  
2236/2255 KB   aenv) 2228/2255 KB  
INFO[0021] (javaenv) 2240/2255 KB  
2252/2255 KB   aenv) 2244/2255 KB  
INFO[0021] (javaenv) 2255/2255 KB  
INFO[0021] (javaenv) Downloaded: https://repo.maven.apache.org/maven2/com/google  
/guava/guava-19.0/guava-19.0.jar (2255 KB at 4922.3 KB/sec)  
INFO[0021] (javaenv)  
INFO[0021] (javaenv)  
INFO[0021] (javaenv)  
INFO[0021] (javaenv)  
INFO[0021] (javaenv)  
INFO[0021] (javaenv)  
INFO[0021] (javaenv) :: Spring Boot :: (v2.1.7.RELEASE)  
INFO[0021] (javaenv) 2019-09-01 18:23:45.418 INFO I --- [ main] com.f  
tools.javaenv.javaEnvApplication : Starting JavaEnvApplication on 8c8483e9566  
2 with PID 1 (/workspace/apps/javaenv/target/classes started by local1000 in /wo  
kworkspace/apps/javaenv)  
INFO[0021] (javaenv) 2019-09-01 18:23:45.420 INFO I --- [ main] com.f  
tools.javaenv.javaEnvApplication : No active profile set, falling back to def  
ault profiles: default  
INFO[0022] (javaenv) 2019-09-01 18:23:45.826 INFO I --- [ main] s.d.  
r.config.ConfigurationDelegate : Bootstrapping Spring Data repositories in
```

Una vez puesto en marcha el servicio. Se mostrarán los endpoint y los métodos de la API de usuarios.



## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

```
// Get All Users
@GetMapping("/users")
public List<User> getAllUsers() {
    return userRepository.findAll();
}

// Create a new User
@PostMapping("/users")
public User createUser(@Valid @RequestBody User user) {
    return userRepository.save(user);
}

// Get a Single USER
@GetMapping("/users/{id}")
public User getUserById(@PathVariable(value = "id") Long userId) throws UserNotFoundException {
    return userRepository.findById(userId)
        .orElseThrow(() -> new UserNotFoundException(userId));
}

// Update a User
@PutMapping("/users/{id}")
public User updateUser(@PathVariable(value = "id") Long userId,
    @Valid @RequestBody User userDetails) throws UserNotFoundException {
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new UserNotFoundException(userId));

    user.setName(userDetails.getName());
    user.setLastName(userDetails.getLastName());
    user.setEmail(userDetails.getEmail());

    User updatedUser = userRepository.save(user);

    return updatedUser;
}

// Delete a User
@DeleteMapping("/users/{id}")
public ResponseEntity<?> deleteUser(@PathVariable(value = "id") Long userId) throws UserNotFoundException {
    User user = userRepository.findById(userId)
        .orElseThrow(() -> new UserNotFoundException(userId));

    userRepository.delete(user);

    return ResponseEntity.ok().build();
}
```

Figura 101: API de usuarios en Java.

Ahora con Postman se lanzará una petición para crear un usuario:

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

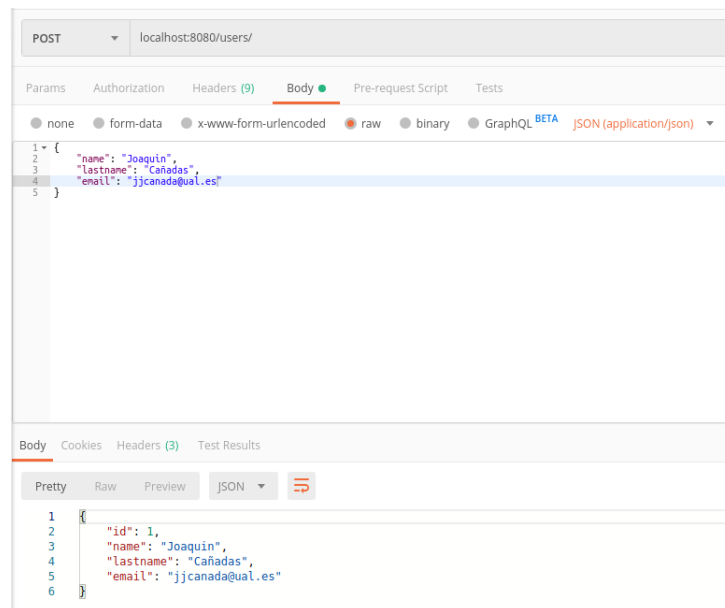


Figura 102: Creación de un usuario con la API de Java.

Comprobando en la base de datos:



Figura 103: Usuario creado en la base de datos.

Finalmente, el usuario aparece en la tabla, garantizando el funcionamiento del servicio `javaenv`.

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

### 5.2.4 Ejemplo API REST en Go con Redis

Del mismo modo que se ha hecho en el punto anterior, se mostrará el funcionamiento de una API hecha en Go, que interacciona con una base de datos NOSQL de Redis.

En primer lugar, se ejecutará el comando ‘ualtools start goenv’:

```
david@galtiplaconsulting ~/escritorio/example ualtools start goenv
INFO[0000] Start service                      service=redis
INFO[0000] Start service                      service=goenv
INFO[0000] (goenv) 18:39:46: prep: go install ./cmd/goenv
INFO[0000] (goenv) >> done (109.269056ms)
INFO[0000] (goenv) 18:39:46: daemon: goenv
INFO[0000] (goenv) >> starting...
INFO[0000] (goenv) time="2019-09-01T18:39:46Z" level=info msg="Listen and serve
in port 8000"
```

Figura 104: Ejecución del servicio de desarrollo goenv.

Una vez arrancado, aquí están los endpoint de la API y los métodos a los que llama, que consisten en un sistema de gestión de nombres asociados a un trabajo:

```
router.HandleFunc("/names/{name}", names.Get).Methods("GET")
router.HandleFunc("/names", names.Add).Methods("POST")
router.HandleFunc("/names/{name}", names.Delete).Methods("DELETE")
```

Figura 105: Ejecución del servicio de desarrollo goenv.

Ahora, se lanzarán 2 peticiones a Postman, una para crear un nombre-trabajo y otra para mostrarlo (ya que no se ha añadido a UALTools una herramienta estilo phpmyadmin para visualizar los registros de Redis):

5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

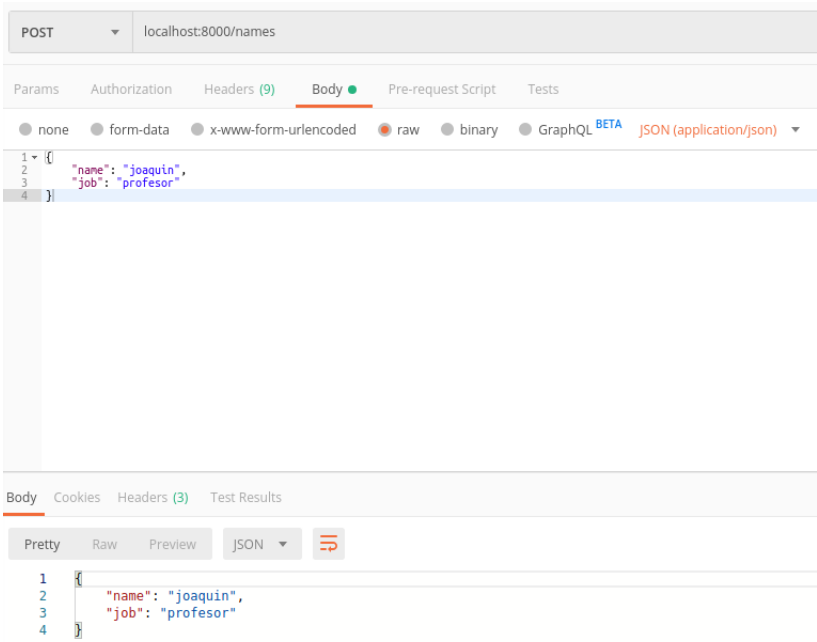


Figura 106: Creación de un par nombre-trabajo.

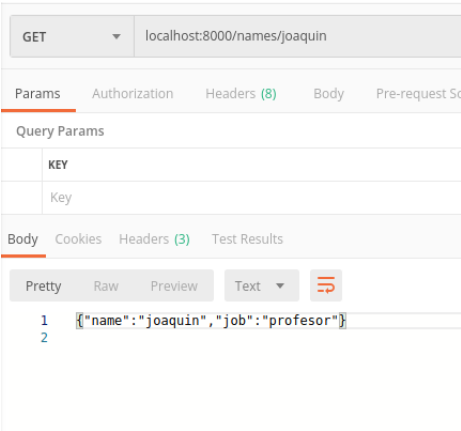


Figura 107: Visualización del par nombre-trabajo creado anteriormente.

Finalmente, se comprueba que el servicio goenv funciona correctamente, y es capaz de interactuar con redis.

## 5.2 Ejecución de un entorno de desarrollo con distintos servicios de UALTools

## 6 Conclusiones

UALTools es finalmente la herramienta que el autor esperaba. Una aplicación capaz de simplificar la instalación de múltiples lenguajes de programación, librerías y otra serie de utilidades a la instalación de una aplicación CLI, que es sencilla de utilizar y puede facilitar el proceso de aprendizaje de un desarrollador novel. Además, es una aplicación que dada su arquitectura es capaz de ampliar su número de herramientas y/o funcionalidades de una forma relativamente sencilla. Para ello, volver a mencionar que es un proyecto público de GitHub abierto a PullRequest.

### 6.1 Trabajo futuro

Se da con este punto por concluido el presente TFG que continuará con un Complemento de Trabajo de Fin de Grado dónde, se explicará todo el proceso de integración continua, automatización de subida de ficheros y control de versión de la aplicación, permitiendo a los usuarios mantener siempre la aplicación actualizada sin necesidad de ejecutar el comando ‘ualtools pull’ y permitiendo al desarrollador no preocuparse por subir todos los ficheros a storage cada vez que haga algún cambio en la aplicación.

## 6.1 Trabajo futuro

## BIBLIOGRAFÍA

### Bibliografía

- [1] Docker. *Docker-compose*. <https://docs.docker.com/compose/>.
- [2] Google. *Google Cloud Platform*. <https://cloud.google.com/>.
- [3] Alan A. A. Donovan & Brian W. Kernighan. *The Go Programming Language (Addison-Wesley Professional Computing Series)*. 2015.
- [4] Mark Lutz. *Learning Python*. 2013.
- [5] Adrian Mouat. *Using Docker: Developing and Deploying Software with Containers*. 2015.
- [6] Jose L. S. P. T. P. T. Krishnan y Ugia Gonzalez. *Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects*. 2015.
- [7] Ben Straub Scott Chacon. *Pro Git*. 2014.
- [8] sirupsen. *logrus*. <https://github.com/sirupsen/logrus/>.
- [9] spf13. *Cobra*. <https://github.com/spf13/cobra>.
- [10] *Sublimetext*. <https://www.sublimetext.com/>.
- [11] *Ubuntu Mate*. <https://ubuntu-mate.org/>.



## BIBLIOGRAFÍA



UALTools es una herramienta de creación de entornos de desarrollo en Go y Java, que permite la ejecución de scripts o test unitarios básicos en Java, Python y Go, permite crear bases de datos en MySQL y redis y que además posibilita el crear bases de datos MySQL a través de la ejecución de migraciones. UALTools es sencillo de instalar (sólo requiere la instalación de Docker) y configurar (se configura con un fichero Llamado ualtools.yml con una sintáxis legible). Proporciona acceso a un set de herramientas que permite ejecutar programas o scripts de prueba de la forma más sencilla posible, configurando únicamente las cosas más esenciales de cada una de las herramientas involucradas en el proceso.

UALTools is a tool for creating development environments in Go and Java, which enables the execution of basic scripts or tests in Java, Python and Go, allows creating databases in MySQL and Redis and also makes it possible to create databases MySQL through the execution of migrations. UALTools is easy to install (only requires the installation of Docker) and configure (it is configured with a file ualtools.yml). It provides access to a set of tools that allows you to run programs or unit tests in the simplest way possible, configuring only essential things of each of the tools involved in the process.